



LIBRARY OF THE  
UNIVERSITY OF ILLINOIS  
AT URBANA-CHAMPAIGN

510.84

I26r

no. 415-420

cop. 2







Digitized by the Internet Archive  
in 2013

<http://archive.org/details/illiaciicompute417nord>



ILLIAC III COMPUTER SYSTEM MANUAL:  
TAXICRINIC PROCESSOR

VOLUME 1

by

Bernard J. Nordmann, Jr.

November 24, 1970



THE LIBRARY OF THE

NOV 9 1972

UNIVERSITY OF ILLINOIS  
AT URBANA-CHAMPAIGN

DEPARTMENT OF COMPUTER SCIENCE  
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS



Report No. 417

ILLIAC III COMPUTER SYSTEM MANUAL:  
TAXICRINIC PROCESSOR\*

VOLUME 1

by

Bernard J. Nordmann, Jr.

November 24, 1970

Department of Computer Science  
University of Illinois  
Urbana, Illinois 61801

This report replaces Report No. 341

\*Supported by Contract AT(11-1)-2118 with the U.S. Atomic Energy Commission.





## ACKNOWLEDGMENT

The conceptual design of the Taxicrinic Processor of the Illiac III Computer System is largely the work of three individuals: Roger E. Wiegel, Bruce H. McCormick and the author. Dr. R. M. Lansford emphasized the importance of the associative addressing scheme wherein the processor can automatically enter the Segment Name Table for additional base descriptors, as needed. This facility, central to an effective Operating System, has been incorporated into the design of the Taxicrinic Processor.

Familiarity with the Illiac III Programming Manual, edited by B. H. McCormick and R. M. Lansford, March 1968 is assumed. Where discrepancies exist, they should be resolved in favor of the present manual as the more authoritative document.

Section III, which discusses the coupling of the Taxicrinic Processor to the Exchange Net, rests heavily upon File No. 790 by Paul S. Krabbe entitled "A Discussion of Illiac III Processor-Unit Communication Via the Exchange Net". Dan E. Atkins also read this section for consistency with the Arithmetic Unit design.

Bruce H. McCormick and Mary Ann Hagen assisted in the arduous task of editing this volume of the TP Manual. Mrs. Donna J. Stutz labored over the many revisions and rewrites to type the document. The illustrations were prepared by John Otten, Eric Storm and Ron Morrison.

The corrections and additions for the second edition were edited by John O'Donnell and typed by Mrs. Betty Gunsalus. The supplementary illustrations were drawn by Stan Zundo and Bill Giblin.



## OUTLINE

### 1. GENERAL INFORMATION

#### 1.1 Report Conventions

1.1.1 Report Organization

1.1.2 Signal Name Conventions

#### 1.2 Brief Overview of Illiac III Computer System

#### 1.3 General Description of the Taxicrinic Processor

1.3.1 Basic Purpose of the Taxicrinic Processors

1.3.2 The General Organization of the Taxicrinic Processor

1.3.2.1 The TP Registers

1.3.2.2 The TP Subprocessors

### 2. SUBPROCESSOR DESCRIPTIONS

#### 2.1 Permuter

2.1.1 Permuter-Functional Description

2.1.1.1 Permuter Input

2.1.1.2 Permuter Control Logic

2.1.1.3 Permutation Decoding

2.1.1.4 Permuter Output

2.1.1.5 Inhibit Generation Logic

2.1.1.6 Gate Inhibits for the LR and IR

2.1.1.7 Permuter Register Gate Drivers

2.1.2 Signal Name Lists for the Permuter

2.1.2.1 Control Signals

2.1.2.2 Internal Signals Used by the Permuter





- 2.1.3 Permuter - PL/1 Description
- 2.1.4 Permuter - Logic Description
  - 2.1.4.1 The Basic Permuter Card
  - 2.1.4.2 Inhibit Generator
  - 2.1.4.3 Permutation Control Logic

## 2.2 Nine Bit/Byte Buffer Storage Blocks

- 2.2.1 Buffer Storage - Functional Description
- 2.2.2 Signal Name List for Buffer Storage
- 2.2.3 Buffer Storage - PL/1 Description
- 2.2.4 Buffer Storage - Logical Description

## 2.3 Operand Stack

- 2.3.1 Operand Stack - Functional Description
  - 2.3.1.1 Operand Stack Control Registers
  - 2.3.1.2 The Constant Generator
  - 2.3.1.3 Byte Selection Logic
  - 2.3.1.4 Overflow-Underflow Sensing Logic
  - 2.3.1.5 5-Bit Adder - Functional Description
- 2.3.2 Signal Name Lists for the Operand Stack
  - 2.3.2.1 Control Signals
  - 2.3.2.2 Internal Signals Used by the Operand Stack
- 2.3.3 Operand Stack - PL/1 Description
- 2.3.4 Operand Stack - Logical Description
  - 2.3.4.1 Operand Stack Storage
  - 2.3.4.2 Overflow-Underflow - Logical Description
  - 2.3.4.3 5-Bit Adder - Logical Description

## 2.4 The Pointer Registers

- 2.4.1 Pointer Register Formats
  - 2.4.1.1 Pointer Stack Format
  - 2.4.1.2 List Processing Format
  - 2.4.1.3 Available Space Format



## 2.4.2 Pointer Registers - Functional Description

### 2.4.2.1 Pointer Register Storage

### 2.4.2.2 Pointer Register Control Registers

### 2.4.2.3 Pointer Register Selection Process

### 2.4.2.4 Tag Setting and Sensing

## 2.4.3 Signal Name Lists for the Pointer Registers

### 2.4.3.1 Control Signals

### 2.4.3.2 Internal Signals Used by the Pointer Registers

## 2.4.4 Pointer Registers - PL/1 Description

## 2.5 Base Registers

### 2.5.1 Base Registers - Function Description

#### 2.5.1.1 Base Register Storage

#### 2.5.1.2 Associative Register Storage

#### 2.5.1.3 Queue Counters

#### 2.5.1.4 Association Logic and Associative Registers

#### 2.5.1.5 Base Register Read/Write Selection

### 2.5.2 Signal Name Lists for the Base Registers

#### 2.5.2.1 Control Signals

#### 2.5.2.2 Internal Signals Used by the Base Registers

### 2.5.3 Base Registers - PL/1 Description

## 2.6 Instruction Buffer Register

### 2.6.1 Instruction Buffer Register - Functional Description

#### 2.6.1.1 Instruction Buffer Register Storage

#### 2.6.1.2 The Instruction Counter and Selection Logic





## 2.6.2 Signal Name Lists for the Instruction Buffer Register

### 2.6.2.1 Control Names

### 2.6.2.2 Internal Names Used by the Instruction Buffer Register

## 2.6.3 Instruction Buffer Register - PL/1 Description

## 2.6.4 Instruction Buffer Register - Logical Description

### 2.6.4.1 Instruction Counter Logic

## 2.7 The 32 Bit Adder

### 2.7.1 32 Bit Adder - Functional Description

#### 2.7.1.1 Block Diagram Description

#### 2.7.1.2 Adder Timing

#### 2.7.1.3 Adder Overflow

### 2.7.2 Signal Name Lists for the Adder

#### 2.7.2.1 Control Signals

#### 2.7.2.2 Internal Signals Used by the Adder

### 2.7.3 32 Bit Adder - PL/1 Description

## 2.8 Boolean/Shift Logic

### 2.8.1 Boolean/Shift Logic - Functional Description

#### 2.8.1.1 Boolean Logic

#### 2.8.1.2 Shift Logic

#### 2.8.1.3 The M-Counter

### 2.8.2 Signal Name Lists for the Boolean/Shift Logic

#### 2.8.2.1 Control Signals

#### 2.8.2.2 Internal Signals Used by the Boolean/Shift Logic

### 2.8.3 Boolean/Shift Logic - PL/1 Description

### 2.8.4 Boolean/Shift Logic - Logical Description

#### 2.8.4.1 Boolean/Shift Logic

#### 2.8.4.2 M-Counter Logic



## 2.9 Algebraic/Logical Compare Logic

2.9.1 A/L Compare Logic - Functional Description

2.9.2 Signal Name Lists for the A/L Compare Logic

2.9.2.1 Control Signals

2.9.2.2 Internal Signals Used by the A/L Compare Logic

2.9.3 A/L Compare Logic - PL/1 Description

## 2.10 Cell Size Generator

2.10.1 Cell Size Generator - Functional Description

2.10.2 Signal Name Lists for Cell Size Generator

2.10.2.1 Control Signals

2.10.2.2 Internal Signals Used by the Cell Size Generator

2.10.3 Cell Size Generator - PL/1 Description





### 3. INTERFACE TO THE OTHER SUBSYSTEMS

#### 3.1 The Exchange Net

3.1.1 INBUS and ØUTBUS

3.1.2 Principal Parts of the Exchange Net

3.1.3 Standard Signals Used in the Control Bytes of all  
Processors and Units

3.1.4 Standard Signal Sequencing (Control Byte)

3.1.5 Exchange Net - TP Interface - General

#### 3.2 The Core Storage Units

3.2.1 Requirements for Processors and the Core Storage Units

3.2.2 Input to the Storage Units

3.2.3 Output from the Storage Units

3.2.4 Processor-Core Storage Unit Signal Sequencing

3.2.5 Individual Differences in the Types of Units

3.2.6 Exchange Net - TP Interface for the Memory Units

#### 3.3 The Arithmetic Units

3.3.1 Input to the Arithmetic Units

3.3.2 Output from the Arithmetic Units

#### 3.4 The Pattern Articulation Unit

#### 3.5 The Interrupt Unit

#### 3.6 The I/O Processors



## 1. GENERAL INFORMATION

### 1.1 Report Conventions

#### 1.1.1 Report Organization

This manual serves as a general information source on the Taxicrinic Processor. The manual will eventually contain all necessary information to enable a person to understand the TP in adequate detail to check out and diagnose the hardware.

Since this manual contains much detailed information, it has been necessary to organize it in a manner such that the reader can obtain needed information with minimum lost time and effort due to reading non-essential information. To this end the manual has been divided into five sections: Introduction, Sub-processor Descriptions, Interface Information, Control Sequence Descriptions and the Logic Directory.

The first section describes the Illiac III System in a general manner and gives a brief description of the Taxicrinic Processor.

The second section gives a detailed description of each sub-processing section in the TP. For each section there is first a summary, then a detailed functional description. Next the input and output signals from and to the other subprocessors are described. Finally for each section there is a detailed logic description complete with normal signal states, detailed boolean descriptions and explanations of unusual design choices.

The third section describes the interface between the TP and the other processors and units in the Illiac III system. A short description of the operation of the Exchange Net and each attached processor and unit is given. The necessary interface signals which must travel back and forth between the processors/units and the TP are then described.

The fourth section describes the control unit sequences. Each sequence is described as to its function and then a detailed description of its flow chart is given. Finally the description of the actual logic is presented. Again, anything unusual in the way of logic is explained in

detail. A discussion of the control logic philosophy is presented as an appendix to this section.

The fifth section, the Logic Directory, summarizes various details of the TP Logic. It contains a complete listing of all signal names used in the TP along with their definitions and places of occurrence. There is also a list of load currents, a list of board counts and similar information.

### 1.1.2 Signal Name Conventions

The signal names used in the Taxicrinic Processor follow a variety of signal name conventions. There are two corresponding sets of signal names: the logic names, which are used in the logical design, and the simulation names, which are used in the PL/1 logical simulation of the TP.

Every logic name, S, can be expressed in two ways, S or  $\bar{S}$ . The S signal name represents the true state (also known as '+' or '1') of the assertion or statement associated with S. The  $\bar{S}$  signal name represents the false state (also known as '-' or '0') of the assertion or statement associated with S. When a line labeled S is in the true state, i.e. it is "on", any lines labeled  $\bar{S}$  must, by definition, be in the false state, i.e. they are "off".

On the logic drawings themselves the fact that a signal is barred or unbarred indicates the normal resting state of the signal line. In other words a signal labeled S will normally be '-', '0', or 'false' and will only become '+', '1', or 'true' when the assertion represented by S becomes true. In the same manner, a signal labeled  $\bar{S}$  will normally be '+', '1', or 'true', and will only go to '-', '0', or 'false' when S becomes true (i.e.  $\bar{S}$  becomes false).

In discussing logic signals the "barred" signal names will be expressed using either an overbar, as in  $\bar{S}$ , or a terminal slash as in S/. These two variants are equivalent. The slash is generally used in computer listings while the overbar is generally used in the text of the TP Manual and in the logic drawings.

Whenever a signal moves from its normal resting state it is said to be active. Thus, to say that S is activated is to imply that all  $\bar{S}$  signals become '0' and all S signals become '1'. This terminology is often used when both versions of the signal appear in the same section of logic.

Two signals are said to be logically equivalent if the activation of either signal implies that the other signal will be active. This is a DC definition, i.e. the states may not exactly coincide in time since there may be a delay between the time of activation of the two signals. Thus DB, DBB and DBP are all logically equivalent signals although there will be several tens of nanoseconds between the time when DBP turns on and the time when DBB turns on.

Some signal names are classified into one of four classes: inhibits, gates, enables or selects. The classifications are in some cases rather arbitrary but in general they follow certain broad outlines. Inhibits generally keep something from happening. Gates, transmit waiting data from one bus or register to another. Enables generally activate some group of logic. Selects usually choose one of several possible actions or blocks of logic to be operated.

As can be seen, these definitions overlap considerably and in many specific instances can be quite arbitrary. At any rate, signal names which have been assigned to one of these classes are designated at the end with a slash followed by one of the letters N (inhibit), G (gate), E (enable), or S (select). This slash and letter appear after the main body of the signal name but before any slash indicating a barred signal. Thus, examples of these signals are DLR1/N/, inhibit DR to LR gate - byte 1, DRP/G, DR to Permuter input gate, ADD/E, 32 bit adder enable, and PR2/S/, Pointer Register 2 select.

The signal names themselves usually contain up to 8 letters counting any slash and N, G, E, or S, but not counting the terminal slash. Signal names in general may not end in the letters I or  $\phi$ . This is to avoid confusion with subscripted signal names with subscripts of one or zero. Unfortunately, this convention is not rigorously followed. In most cases the exceptions have not been changed simply because of historical precedent. One example of this is the FBI bus. The signal FBI17 is often confused as FB<sub>117</sub> instead of FBI<sub>17</sub>.

As shown above, logic signal names are subscripted by adding numbers at the end of the name. If the signal name is classified using the slash and letter convention, the subscript appears before the slash (i.e. DLRL/N). Double subscripts cannot be handled by the current wiring list generating program and are therefore frowned upon. However, they can be used by inserting the first subscript in the body of the signal name, i.e. T2R1 is temporary register 2, bit 1 and N6R3 is name register 6, bit 3. This allows the wiring program to assume that there are 15 signal names, NORi through N14Ri, and it treats them as any other single subscript signal name. There are two exceptions to the double subscript rule in the TP logic, IBPL13 and IBPL21. These are, once again, historical exceptions and stand for 2 operations each, i.e. inhibit byte 1 permute left 3, and inhibit byte 2, permute left 1.

In some cases the loading on a given signal name is too large even for a logic driver circuit. In such cases one alternative is to use two drivers with identical inputs and to send the output of one driver to half of the loads and the output of the other driver to the other half. However, since these signals are not electrically identical they need to have different names. One way to do this is to add different numbers at the end of each name. Thus PSNW5/S1 and PSNW5/S2 both indicate that PR Segment Name Register #5 has been selected for writing.

The PL/1 logic simulator for the TP generally uses the same signal names as the TP logic. However, there are several modifications due to the nature of the simulation and the allowable names in PL/1.

First of all, since slashes are not allowed in PL/1 names, all of the slashes in the names in the TP simulation are omitted. Thus DRP/G becomes DRPG in the simulation.

Secondly, subscripted logic names in the TP logic are represented using the standard subscript notation in PL/1. This means that DB17 becomes



DB (17) in PL/1, and DBD2/G becomes DBDG(2). Double subscripts may be expressed either as a whole block of related names with single subscripts (i.e. T1R1 and T2R1 become T1R(1) and T2R(1)) or as doubly subscripted PL/1 variables (i.e. N6R3 becomes NR(6, 3)). The type of representation chosen depends on how much it simplifies the PL/1 code.

Probably the most important difference between the logic signal names and the simulator signal names is the lack of barred control signals in the simulator. The simulator always uses the unbarred version of control signals. Thus if a **barred signal** is to be simulated the simulator will use an unbarred signal and test for it being '1' instead of using a barred signal and testing for '0'. This distinction can be maintained if it is always remembered that when a simulation signal name is '1' the corresponding logic signal names are activated and thus the unbarred forms of the signal are '1' and the barred forms of the signal are '0'.



## 1.2 Brief Overview of the Illiac III Computer System

The Illinois Pattern Recognition Computer, Illiac III, is a digital processor for visual information. It is primarily designed for automatic scanning and analysis of massive amounts of relatively homogeneous visual data. In particular the design is an outgrowth of studies at this laboratory of a computer system capable of scanning, measuring and analyzing in excess of  $3 \times 10^6$  bubble chamber negatives per year.

Illiac III, though specifically designed to process visual information, also provides complete facilities for standard general-purpose computation. Both the picture processing and general-purpose computation facilities of Illiac III will be available to users on a time-sharing basis.

As can be seen in Figure 1.2, Illiac III is a multi-processor computer system. Six processors (4 Taxicrinc Processors and 2 Input/Output Processors) access in parallel the computational/storage units consisting of 2 Arithmetic Units, 1 Interrupt Unit, 1 Pattern Articulation Unit, and 4 Storage Units. Each computational/storage unit of the computer system specializes in a particular activity. Thus, for example, all floating-point computation is done in the Arithmetic Units, while picture processing is performed primarily by the Pattern Articulation Unit. Processors, on the other hand, analyze user jobs and route their constituent tasks to the appropriate specialized processing units. The individual processors of the system can operate simultaneously and independently (within the limits imposed by the Operating System) with a consequent increase in overall efficiency.

The Input/Output Processors (IØP) are attached via Channel Interface Units and Device Controllers to various input and output devices. Among facilities important for the ingestion of visual information are 8 CRT

flying spot scanners: two for 70 mm film, two for 46 mm film, two for microfilm/microfiche, and two for microscope slides. These scanners can also operate as film cameras and thus serve as both input and output devices. Monitor stations have also been attached to the Input/Output system. These each consist of a CRT display, a typewriter, and a magnetic tape unit; and are provided to assist human control of the analysis.

The duty of the Pattern Articulation Unit (PAU) is to perform local preprocessing on the input from the scanners, such as track thinning gap filling, line element recognition, etc. The logical design of this all-digital processor has been optimized for the idealization of the input image to a line drawing. Nodes representing end points, points of inflection, points of intersection, etc. are labeled in parallel by appropriate programming under overall control of the Taxicrinic Processor. The abstract graph describing the interconnection of labeled nodes is then extracted as a list structure, which comprises the normal output of the Pattern Articulation Unit.

This output is then operated on by a Taxicrinic Processor (TP), which assembles such graphs into coherent list structures subject to a recognition grammar and then syntactically categorizes them to complete the visual recognition process. The Taxicrinic Processors are primarily responsible for the execution of user programs, that is, to oversee the operations of the Pattern Articulation Unit, the Arithmetic Unit and to initiate input/output operations in the IOP's by making requests to the Interrupt Unit.

The Arithmetic Unit (AU) is used exclusively for performing arithmetic operations for the TP. Although there are a few simple arithmetic operations which can be done in a TP (e.g., integer addition) the more complicated operations are done in the AU. The AU has been optimized for double-word floating point arithmetic.

The Interrupt Unit (IU) handles all the interrupt requests from the TP and IOP. When an interrupt is requested it notifies the proper processors which then take appropriate action.

All of the Illiac III processors and units communicate with each other through the Exchange Net (XN) as shown in Figure 1.2. The Exchange Net is responsible for all the necessary queueing and priority checking.

As noted above, there is indeed a reason for calling one piece of equipment a processor and another a unit, even though the type of operations they perform may both appear to be "processing" operations. In the Illiac III system all major modules are designated as either "processors" or "units" according to their position in the Exchange Net. In Figure 1.2, the processors are shown at the top and bottom and the units are shown on the right. The effect of this division is that processors may communicate directly with units and vice versa but may not communicate directly with each other. If a processor needs to communicate with another processor it must get help from a unit (normally the Interrupt Unit) and if a unit (say the PAU) wants to communicate with another unit (say a storage unit) the information must be transferred through a processor (the TP in this case).

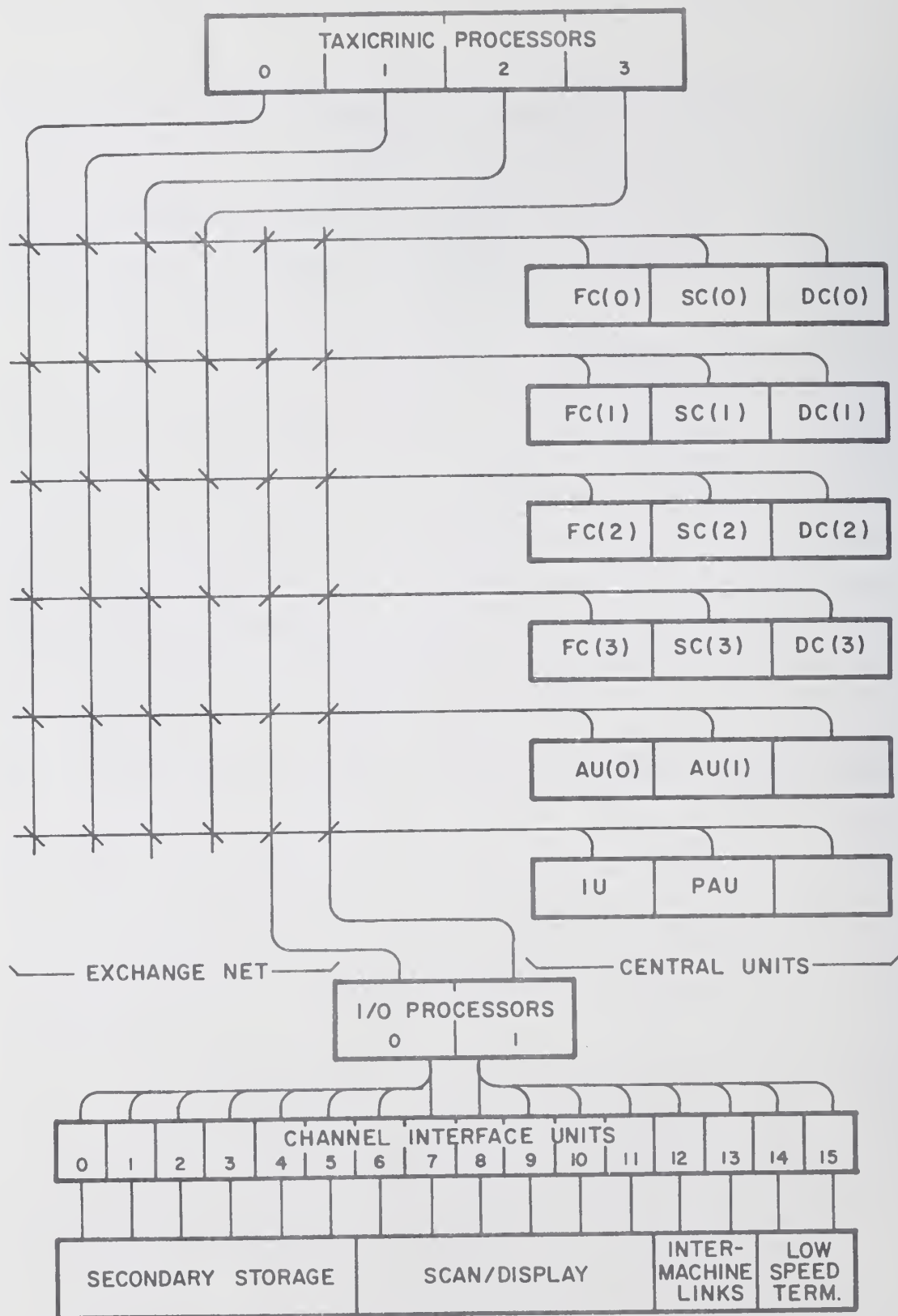


FIGURE 1.2 SCHEMATIC OF ILLIAC III COMPUTER

### 1.3 General Description of the Taxicrinic Processors

#### 1.3.1 Basic Purpose of the Taxicrinic Processors

The Taxicrinic Processors are the central control units of Illiac III. Their principal activities are the manipulation, search and systemization of abstract graphs (bilateral list structures) which have been produced from the pictorial input to the Pattern Articulation Unit. In addition to this they also issue commands to the other Illiac III processors (e.g., Arithmetic Units) whenever they are needed to perform specialized operations. The name "Taxicrinic" comes from two Greek words: ταξις meaning "arrangement" or "pattern", and κρισις meaning to "judge", thus indicating the TP's general purpose, which is to syntactically analyze digitized pictures and other material which can be cast into the form of a directed, labeled graph.

Because the main purpose of a TP is to process list structures, its main operations are centered around this type of operation. A certain number of simple arithmetic instructions can be performed, but, in general, whenever a complicated arithmetic computation must be performed, an Arithmetic Unit (AU) is utilized. This is done by giving an AU a command via the Exchange Net (XN).

The TP also has the ability to initiate input-output operations by giving commands to the Input/Output Processor (IOP) via the Interrupt Unit (IU) and to control the jobs performed by the Pattern Articulation Unit (PAU). The commands necessary to exercise this control are given to the various units concerned through the Exchange Net system which sets up the necessary priority schemes. The Exchange Net also controls the communication between all the processors (including the TP's) and the various storage modules.

It should be kept in mind that all four TP's are exactly the same. Each is equipped with its own private fast registers and operates completely independently of the other TP's.



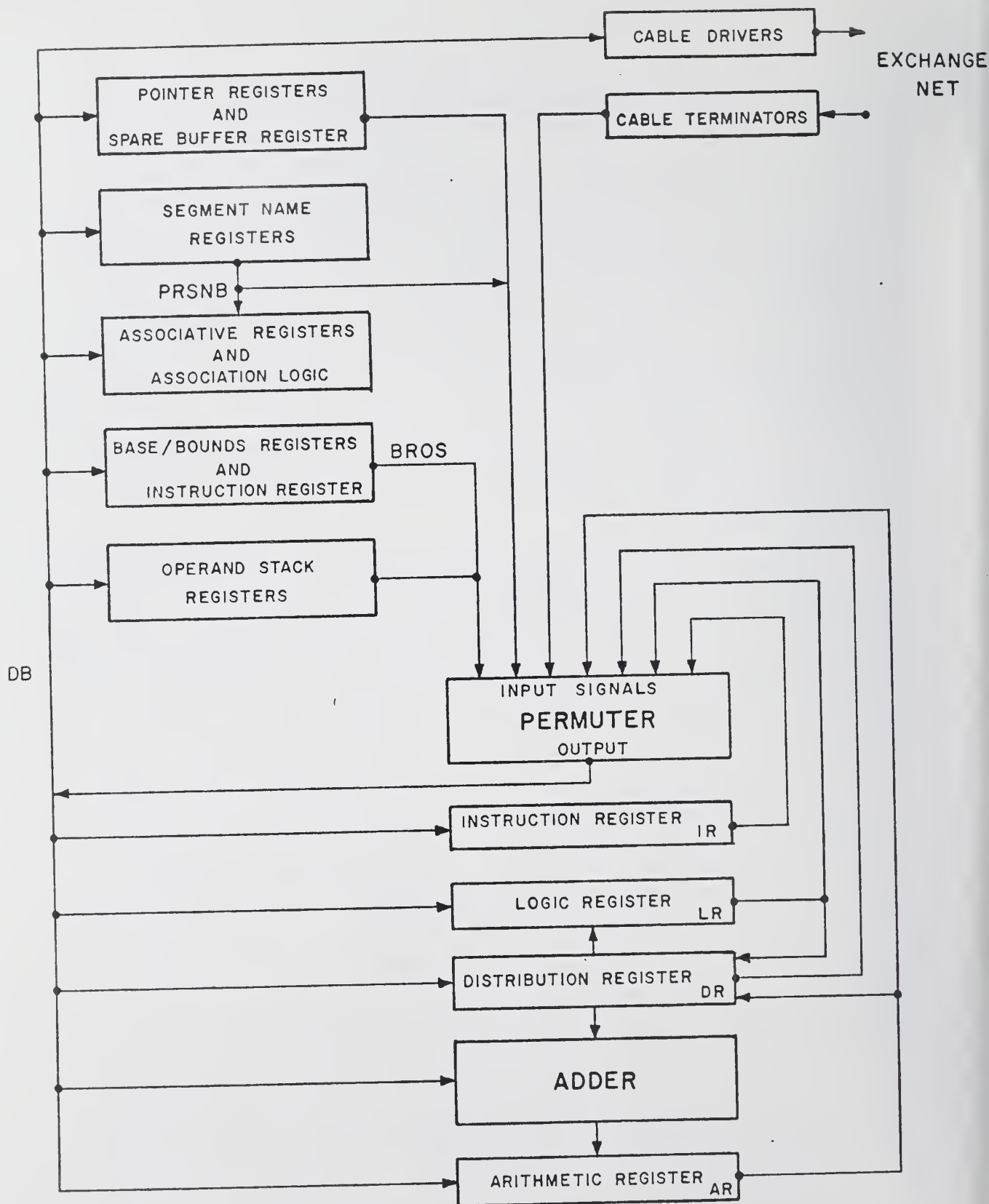


Figure 1.3.1 - Major TP Registers and Data Paths

### 1.3.2 General Organization of the Taxicrinic Processor

#### 1.3.2.1 The Taxicrinic Processor

Figure 1.3.1 shows the main registers and subprocessors of the TP, exclusive of the main control, together with the various data paths between them. The main "active" registers are the Instruction Register (IR), the Logic Register (LR), the Distribution Register (DR), the Arithmetic Register (AR), and the Spare Buffer Register (SBR). These are all 36 bits long (4 bytes of 8 bits and 1 flag each) and are used to hold data temporarily as various instructions are being processed. None of these are directly accessible to the programmer.

The main "storage" registers are the Base Registers (8 registers of 27 bits or 3 bytes), the Associative Registers (7 registers of 16 bits), the Pointer Registers (15 registers of 36 bits or 4 bytes), the PR Segment Name Registers (16 registers of 16 bits), the Operand Stack (one continuous register of 32 bytes) and the Instruction Buffer Register (one double word 8 bytes). These registers contain data which may be used by the programmer during the execution of his program. They are all accessible by programming except for the Associative Registers and the Instruction Buffer Register, although the Base Register may only be changed by "privileged" instructions.

There are other registers in the TP which are not accessible to the programmer and which are used during special operations. These will be fully explained in their respective sections in this manual.

Finally in connection with Figure 1.3.1 it should be noted that the output from the Permuter, the Distribution Bus (DB), is not a register. It consists of 36 lines which serve as inputs to nearly all of the registers and control sections in the TP. The DB is the main avenue of information transfer in the TP.





### 1.3.2.2 The TP Subprocessors

The general structure of the TP is divided up into various sub-processors which are in turn controlled by the TP main control sequencing. These subprocessors communicate with the main control by means of "control lines" which give commands to the subprocessors and "output lines" which give the main control information about the status of the subprocess. The subprocessors are in general fairly independent and need only receive a small number of control line inputs to perform rather complex operations.

The main subprocessor groups are the Permuter, the Operand Stack and Operand Stack Control, the Pointer Registers and Pointer Register Control, the Base Registers and Base Register Control, the Instruction Buffer Register, the 32 Bit Adder, the Boolean/Shift Logic, Algebraic/Logical Compare Logic, and the Cell Size Generator.

The permuter is a system of gates and drivers which is central to the transfer of information in the Taxicrinic Processor. As can be seen in Figure 1.3.1 it is capable of transferring information to and from all the subprocessors of the TP and essentially all of the fast storage registers. One of the reasons that the permuter is required is that operands are not constrained to be on "natural boundaries". For example, a double word operand may have 3 bytes in one double word of core and 5 bytes in the next double word of core. The permuter is used to assemble these bytes into a single double word operand in the TP fast memory.

The name "permuter" is derived from the fact that an input cell may be permuted to make its boundaries coincide with its destination boundaries. The permutation may be likened to a circular left shift on a byte basis. The position of the four bytes is changed, but the position of one relative to the other is not. For example, the cell: A B C D appears as B C D A when permuted left one byte and as D A B C when

permuted left three bytes. Cells smaller than 4 bytes (bytes or halfwords) are assumed to have garbage in the non-data bytes while double words must be permuted one word at a time.

In addition to permuting the input cell, the permuter has the capability to inhibit all of the output bytes in any combination, or any combination of bits in the rightmost output byte (byte 3). This facility enables the permuter, among other things, to generate constants to put on the distribution line (+32, +8, +2 and +1 are provided for) and to mask in bits to the low order positions of, say the Pointer Registers, as they are gated through the permuter.

The actual Permuter logic cards are also used to construct the storage space for the LR, IR, AR and DR. The Permuter is described in Section 2.1.

The Subprocessors for the Operand Stack, the Pointer Register and the Base Register all require a more detailed description than could possibly be given at this time; their complete description will be deferred until Section 2.

The Instruction Buffer Register (IBR) is an eight byte (double word) register used to store that portion of the instruction code which has yet to be processed. It may also be used to store successive instructions which will be executed after the present one.

The 32 bit Adder in the Taxicrinic Processor is used to perform binary addition within the TP. It is also used to generate the outputs for the Boolean operations, EQV and XOR. The adder uses 2's complement number representation and employs two levels of carry lookahead to hasten carry propagation.\* The adder is broken down into eight four-bit sum groups with full carry lookahead within the groups. The second level of lookahead occurs between groups, with lookahead between groups 1, 2 and 3 and between groups 4 through 8. The final carry between these two second level groups is a "ripple" carry.

---

\*For a more complete discussion of carry lookahead adders, see Wiegel, Roger E., "Methods of Binary Addition", DCS Report No. 195, February 1966.

The Boolean logic performs the Boolean functions "AND", "OR", "XOR" and "EQV". The "XOR" and "EQV" functions are generated using part of the 32 bit adder.

The Shift logic employs the permuter to make shifts in multiples of 8 bits. The shift control first makes the highest multiple-of-8 bit shift that it can without exceeding the desired shift. This is done using the permuter and inhibiting the necessary bytes. After this has been done the shift control shifts one bit at a time until the proper number of additional bits have been shifted. Since the flags are not touched in the bit shifting, the shift control only will shift flags in multiples of 8 (i.e., when the permuter is used).

The Algebraic/Logical Compare Logic is used to accomplish algebraic and logical compares (greater, equal and less), the AR equal to zero test, and flag match compares. The instructions realized are CPRA, CPRL, TEST(M), SCAN(M), and TA. Operands are compared by subtracting one from another; then signs are compared and appropriate indicators are set. The logic discussed here can only be used with long or short fixed point numbers; decimal and floating point numbers are routed to the floating point arithmetic unit.



## 2. SUBPROCESSOR DESCRIPTIONS

### 2.1 Permuter

The Permuter is a system of gates and drivers which are central to the transfer of information in the Taxicrinic Processor. As can be seen in Figure 1.3.1, it is capable of transferring information to and from all the subprocessors of the TP and almost all fast storage registers.

The name "permuter" derives from the fact that an input cell may be permuted to make its boundaries coincide with its destination boundaries. A permutation begins with a circular left shift on a byte basis. The position of the four bytes is changed, but the cyclic ordering of the bytes is preserved. For example, the cell:

A	B	C	D
---	---	---	---

appears as

B	C	D	A
---	---	---	---

when permuted left one byte and as

D	A	B	C
---	---	---	---

when permuted left three bytes. Cells smaller than 4 bytes (that is, bytes and half-words) are assumed to have garbage in the non-data positions, while double words must be permuted one word at a time.

In addition to permuting the input cell, the permuter has the capability to inhibit all of the output bytes in any combination, or in the rightmost output byte (byte 3) to inhibit any combination of bits. This

facility enables the permuter, among other things, to generate constants to put on the distribution bus (+32, +8, and +2 are provided for) and to mask in bits to the low order positions of (for example) the pointer registers as they are gated through the permuter. The SCR (Stack Control Register) and OSTR (Operand Stack Top Register) of the Operand Stack and the ICTR (Instruction Counter) are the registers which may be masked in.

Finally it should be noted that the permuter logic cards also physically contain the registers for the LR, DR, AR and IR. Although these registers are not directly used by the permuter to execute its permutation operations, these registers are commonly used as sources or destinations for data (see Figure 1.3.1 ) For example, the permuter boards are used in data transfers from the LR and DR even though this data path is not shown as going through the permuting mechanism. As a matter of fact, since data going from the LR to the DR is not always permuted, the LR can be simply gated directly to the DR whenever this is desired - thus saving time.

## 2.1.1 Permuter - Functional Description

### 2.1.1.1 Permuter Input

The logic used to realize the permuter input gating is as follows:

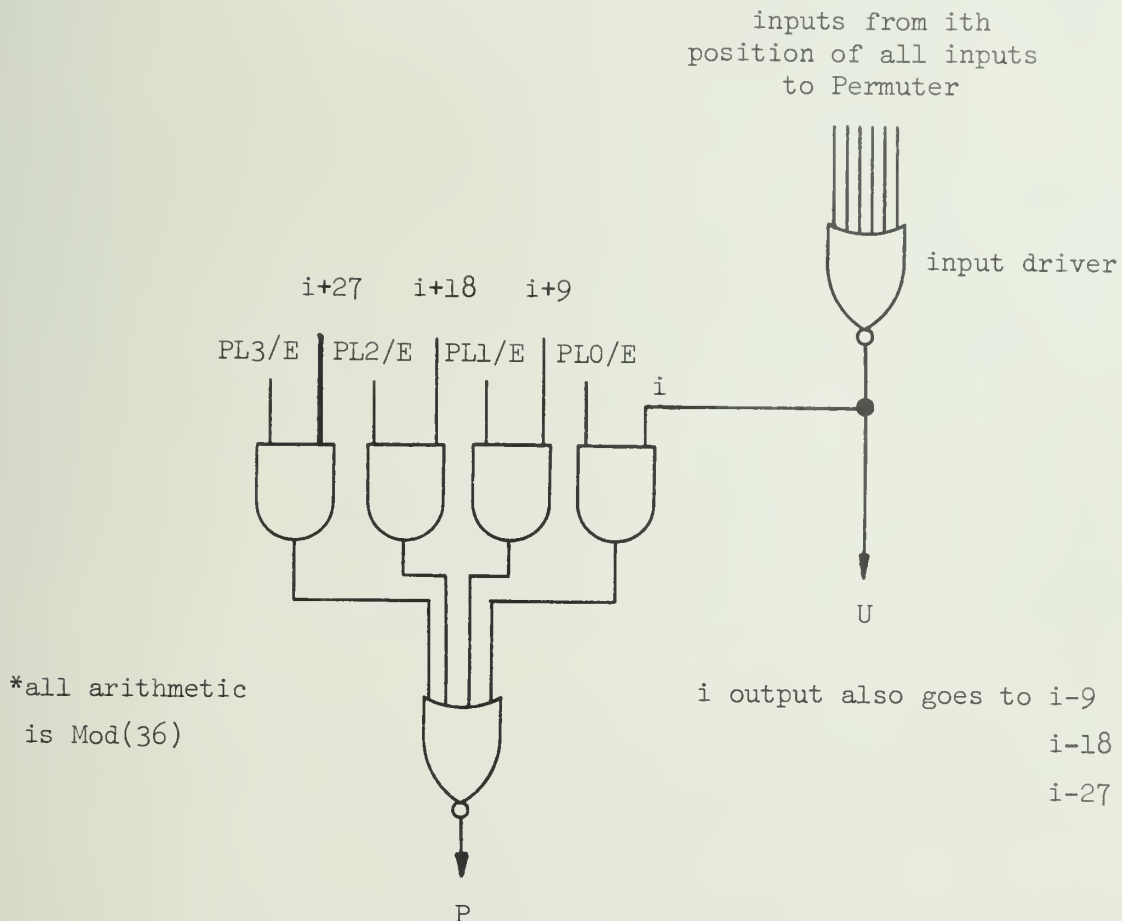


Figure 2.1.1.1/1 - Permuter Schematic

The  $PLi/E$  are the permutation selection gate signals.  $PL3/E$ , for example, enables the left 3 permutation.  $PL0/E$  is the straight-through (left zero) gate.



Every possible input to the permuter has a gated line which is an input to the input driver (see figure above). Thus each bit in any input byte will go to one of the 36 input drivers depending on which byte it is in and its position in that byte. The outputs of these input drivers go to the corresponding bit positions in each of the four byte positions. Here they are gated with the corresponding permute signal.

As an example, input driver 36 will have as inputs the 36th bit of all permuter input registers. This is the 9th position (flag) in the 3rd byte (bytes are numbered from the left 0 to 3). Thus the output of input driver 36 is gated to position 36 by the PL0/E signal, to position 27 by the PL1/E signal, to position 18 by the PL2/E signal and to position 9 by the PL3/E signal (i.e., the 9th bit position of each byte). At the same time position 36 of the permuter will be receiving the outputs of input drivers 9, 18 and 27 which will be gated with PL1/E, PL2/E and PL3/E, respectively. Thus the output (pin P in the diagram above) for the 36th bit position will depend on which of the 4 permute signals is activated.



#### 2.1.1.2 Permutation Control Logic

Once the input has been selected by enabling the gate from the desired register to the input drivers, the Permutation Control Logic ensures that the proper permutation takes place. This logic has 3 inputs: a source bit-pair, a destination bit-pair and a reversing bit.

To determine the permutation which is desired, the position of the first byte of the cell must be known for both the source and destination registers. These positions are given by the bit-pair "addresses" as shown in the figure below. Here an address of "00" indicates that a cell is to be left-justified in its source (destination) register. This address code is also just the low order two bits of a memory address or the low order two bits of the Stack Control Register in the Operand Stack logic or the Instruction Byte Counter.

00	01	10	11
----	----	----	----

Bit-Pair Address Convention

In summary then, the source bit-pair indicates the position of the left-most byte of the cell in its source register. In like manner the destination bit pair gives the position the left-most byte is to occupy in the destination register.

The reversing bit is used to specify the direction in which the data is to flow. In the permuter the paths which must be provided with permutation are:

1. Memory to registers (LR, DR, AR, IR) - left justified
2. Operand Stack to registers - left or right justified

3. Base Registers to registers - left justified
4. Instruction Buffer to registers - left justified  
or justified on byte 1.

The "natural" flow of information is taken to be from the various types of storage (memory, stack, buffer, etc.) to the working registers (LR, DR, AR, IR). For this direction the reversing bit is considered off (0). When the reversing bit is on (1), the data flows in the reverse direction and the permutation is changed accordingly.

Gating signals from main control select the permuter input and signify left or right justification (if required). The Permutation Control Logic then works by feeding the proper bit-pairs into the source and destination pair buses. After the two bit-pairs have been loaded, the reversing bit (RSD/E) is used to gate them to the decoding circuits (see Section 2.1.1.3). If the data flow is in the "natural" direction the source bit-pair and destination bit-pairs are gated straight through to the X and Y busses respectively (as they are called in Section 2.1.1.3). While if the data flow is in the "reverse" direction the source bit-pair is gated to the Y lines while the destination bit-pair is gated to the X lines.

When none of the four "natural" or "reverse" data paths are being evoked, the source and destination inputs to the decoder are in the rest state, 00/00 which decodes as "gate straight through", i.e. without permutation. Thus if a data path other than one of these four is used, and no permutation is necessary, the data transfer can be carried out without concern for the permuter control inputs. One special case does arise: right and left shifting in steps of 9 bits; in this case all the internal permutation signals of the permutation control logic are disabled using the LRS/E signal. The proper permutation for the shift required is decoded (along with inhibit signals -- see Section 2.8.1.2 in the Shift Control Logic) and is routed directly to the permutation gate drivers (output signals PL0, PL1, etc.) to effect the permutation.

The LRS/E signal is also turned on automatically whenever one of the direct permutation control lines from the main control logic is turned on. This inhibits the normal operation of the permutation control and allows the main control logic to determine the permutation.



### 2.1.1.3 Permutation Decoding

If we let  $X_1X_2$  be the source byte "address" bit-pair and  $Y_1Y_2$  be the destination bit-pair, then these can be considered a four-bit binary number which can be decoded to give the proper permutation. If the four permutation control signals are designated PL0, PL1, PL2 and PL3, they can be expressed in terms of  $X_1X_2$  and  $Y_1Y_2$  as:

$$\begin{array}{lcll} \text{Left 0:} & \text{PL0:} = & \overline{X_1}\overline{X_2}\overline{Y_1}\overline{Y_2} & \vee \overline{X_1}X_2\overline{Y_1}Y_2 & \vee X_1\overline{X_2}Y_1\overline{Y_2} & \vee X_1X_2Y_1Y_2 \\ & & 0 & 5 & 10 & 15 \end{array}$$

$$\begin{array}{lcll} \text{Left 1:} & \text{PL1:} = & \overline{X_1}X_2\overline{Y_1}\overline{Y_2} & \vee X_1\overline{X_2}\overline{Y_1}Y_2 & \vee X_1X_2Y_1\overline{Y_2} & \vee \overline{X_1}\overline{X_2}Y_1Y_2 \\ & & 4 & 9 & 14 & 3 \end{array}$$

$$\begin{array}{lcll} \text{Left 2:} & \text{PL2:} = & X_1\overline{X_2}\overline{Y_1}\overline{Y_2} & \vee X_1X_2\overline{Y_1}Y_2 & \vee \overline{X_1}\overline{X_2}Y_1\overline{Y_2} & \vee \overline{X_1}X_2Y_1Y_2 \\ & & 8 & 13 & 2 & 7 \end{array}$$

$$\begin{array}{lcll} \text{Left 3:} & \text{PL3:} = & X_1X_2\overline{Y_1}\overline{Y_2} & \vee \overline{X_1}\overline{X_2}\overline{Y_1}Y_2 & \vee \overline{X_1}X_2Y_1\overline{Y_2} & \vee X_1\overline{X_2}Y_1Y_2 \\ & & 12 & 1 & 6 & 11 \end{array}$$

The PLi signals are then used to drive the four groups of (36 each) permutation gates.

There are also "direct" inputs to the PLi drivers. In this case the desired permutation can be selected directly without decoding from  $X_1$ , etc. An example of this is the shift control logic. If a "direct" input is used, the decoder is inhibited by the LRS/E signal input. The direct inputs go directly to the PLi drivers in complement form.



#### 2.1.1.4 Permuter Output

The basic output of the Permuter is the Distribution Bus (DB). This bus is a 4 byte (36 bit) bus and serves as an input to all the major registers and storage areas in the Taxicrinic Processor.

Although not shown in Figure 1.3.1, the DB is, in reality, made up of two separate busses which are arranged as shown below:

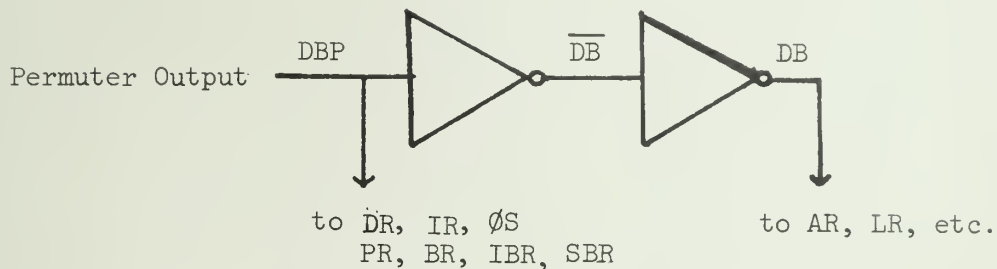


Figure 2.1.1.4/1 - Distribution Bus

The DBP bus specifically drives the DR, IR, ØS, the PR's, the BR's, the IBR and the SBR. The DB bus drives the AR, LR, the NR's, the ØS control logic, the BR name storage flip-flops, the instruction byte counter, the shift boolean logic, the adder input gates, the bounds check logic, and the cable drivers to the Exchange Net.

In this manual, reference to the Distribution Bus (or DB) will mean either of these two busses since their contents are logically equivalent. Reference to DBP however will specifically mean the DBP part of the Distribution Bus.





### 2.1.1.5 Inhibit Generation Logic

In Figure 2.1.1.5/1 we give a more complete version of the logic circuit shown in Figure 2.1.1.1/1. As can be seen, several masking options have been dot-or'ed with the input driver and an inhibit signal has been added to the output. The purpose of this section is to describe the effects which can be achieved using these supplementary features.

First, however, we shall run through a general description of the circuit with particular attention to inhibit conditions. Note that due to the dot-or on the  $\overline{\text{FBI}}$  line, if any of the masking option lines are selected, the  $\overline{\text{FBI}}$  will be forced to "0". This in turn will force the corresponding DBP bit position to "1" (assume for the moment that only PLO/E is "1"). It should be noted that the inhibit line will force the DBP to "0" if it is turned on, independent of the  $\overline{\text{FBI}}$  signal. Thus we have two methods for changing the DBP: We can force the DBP to "1" by turning on one of the masking options, or we can force the DBP to "0" by turning on the inhibit.

Now we can begin to consider the various effects that can be accomplished using these two methods. The permuter is capable of inhibiting any combination of bytes in the output using the IBO/E, IB1/E, IB2/E, and IB3/E signals. Each of these signals is hooked up to the 9 corresponding bit position inhibit signals on the output of the Permuter. In addition any combination of bits in the rightmost byte of the Permuter Output (byte 3) may be inhibited since the bit inhibit lines for this byte are made so they can be controlled individually if desired (see TP logic drawings 04-5 and 04-3).

It is also possible to construct constants using the Permuter. If all of the input gates are turned off (referring again to Figure 2.1.1.5/1), the  $\overline{\text{FBI}}$  lines will all be "1" which will cause the DBP bus to contain all zeroes. Then in order to make positive constants, the corresponding  $\overline{\text{FBI}}$  lines need only have one of their "masking options" turned on. The positive constants which are provided are +1, +2, +8 and +32 (all 2's complement).

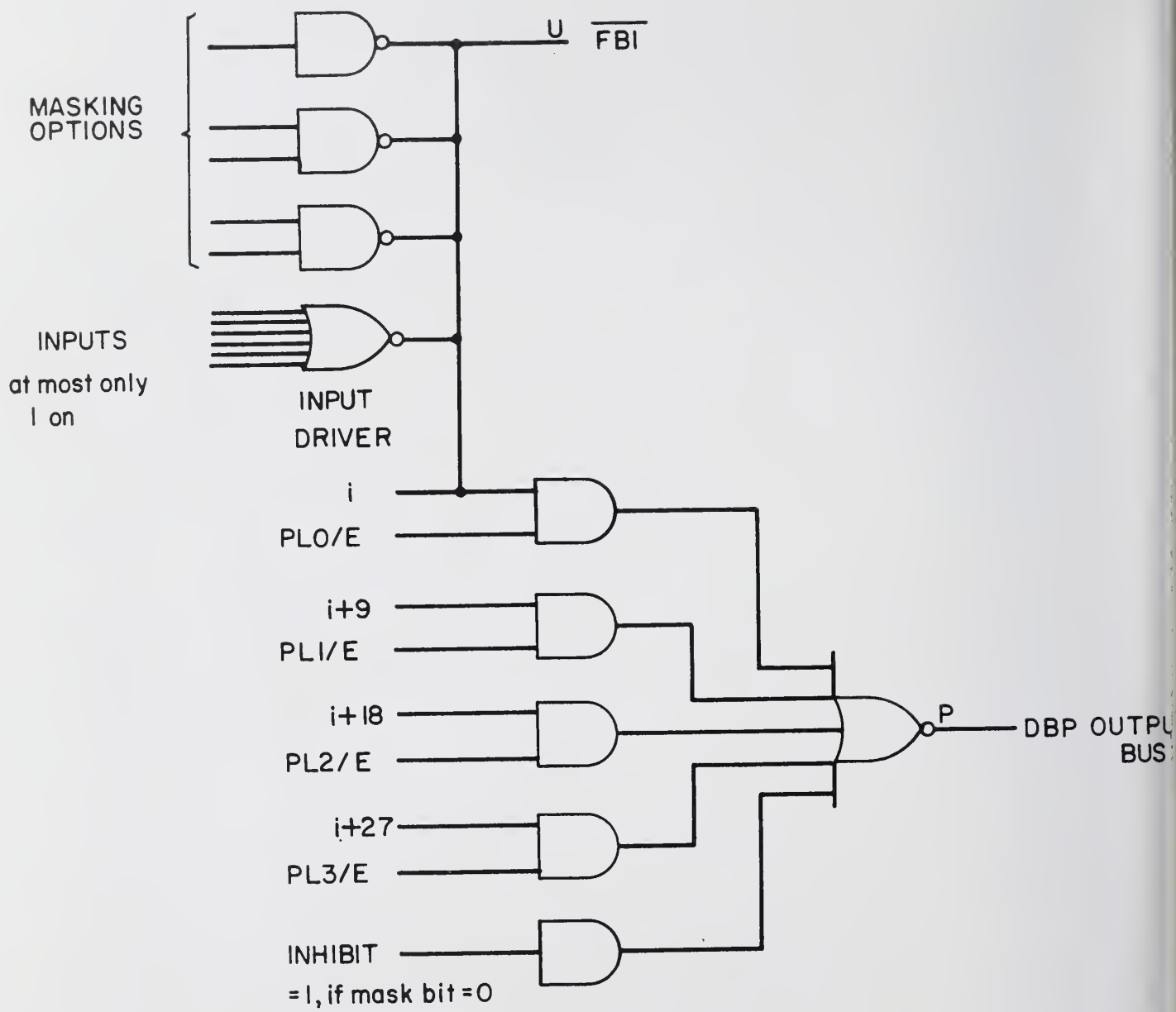


FIGURE 2.1.1.5/1 INHIBIT GENERATION LOGIC.

In order to generate negative constants all 1's are put on the output bus by inhibiting the PLO/E signal with LRS/E. If no other PLi/E signals are on, the DBP will become all ones. Then a one's complement number can be produced by inhibiting the proper output position. This number is changed to two's complement in the adder by injecting a low-order carry. The negative constants which are produced in this manner are -1, -2, -8 and -32 (all 1's complement).

The final effect which can be produced is masking. In many cases it is desired to mask the SCR, ØSTR, or ICT into the lower order bits of a pointer register. Here the  $\overline{\text{FBI}}$  lines corresponding to the positions which are to be masked in, are set to zero by one of the masking option signals. This causes the corresponding output bit position to become "1". The actual contents of the register to be masked is used to set the inhibit lines. If a given bit position in the register is "0", the corresponding inhibit line is set, thus causing the Permuter output to be "0" in that position.

In two of the masking cases given above (ICT and SCR) the masking is done automatically. Whenever PR#0 is used as the input to the Permuter the ICT is automatically masked into the lowest 3 bits. Whenever PR#13 is the input, the SCR is masked into the lowest 5 bits. During Operand Stack overflow or underflow (see Sections 2.2.2.5 and 4.2.2.8) the ØSTR (2 bits) must be masked into the 4th and 5th lowest order bits with zeros in the lowest 3 bits of PR#13. Since the SCR is normally masked into PR#13 there must be an inhibit to SCR masking when the ØSTR is being used instead.

With this in mind the control logic necessary for the automatic masking of the ICT and SCR are given in Figure 2.1.1.5/2 and /3 respectively. The signal definitions are given below:

PRP/G = 1 if a PR is the permuter input  
N13S = 1 if the TGR = 13  
NOS = 1 if the TGR = 0  
NPR13/S = 1 if control selects PR#13  
NPR0/S = 1 if control selects PR#0  
OSTP/G = 1 if the ØSTR is to be masked into PR#13  
OSR = data bus to the inhibit signals  
SCRPG, ICTPG = mask select signals

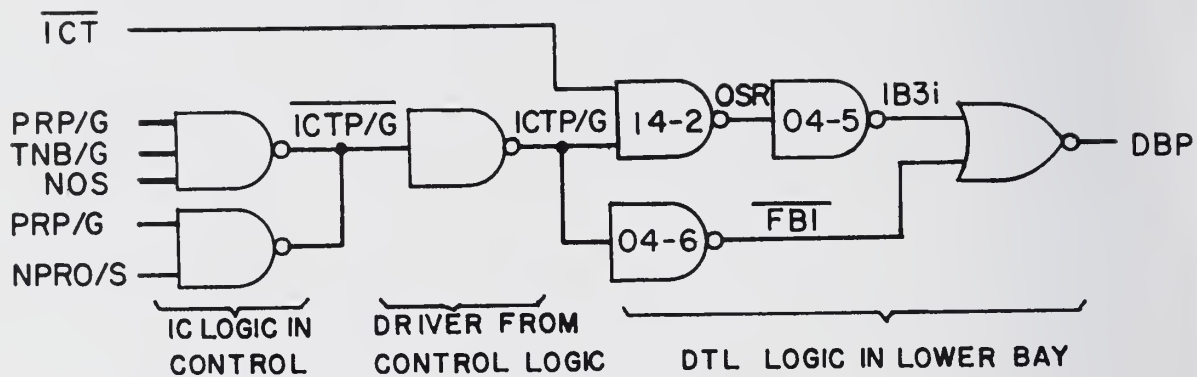


FIGURE 2.1.1.5/2 ICT MASKING CONTROL LOGIC

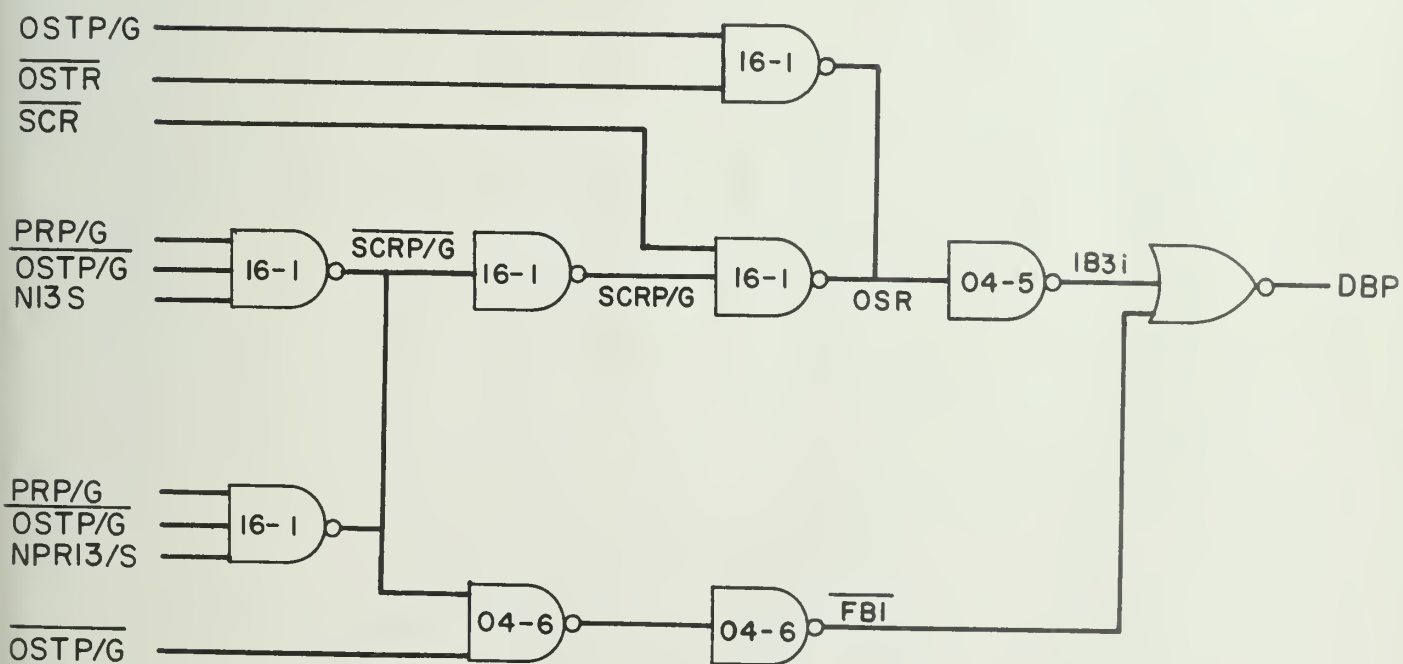


FIGURE 2.1.1.5/3 SCR MASKING CONTROL LOGIC



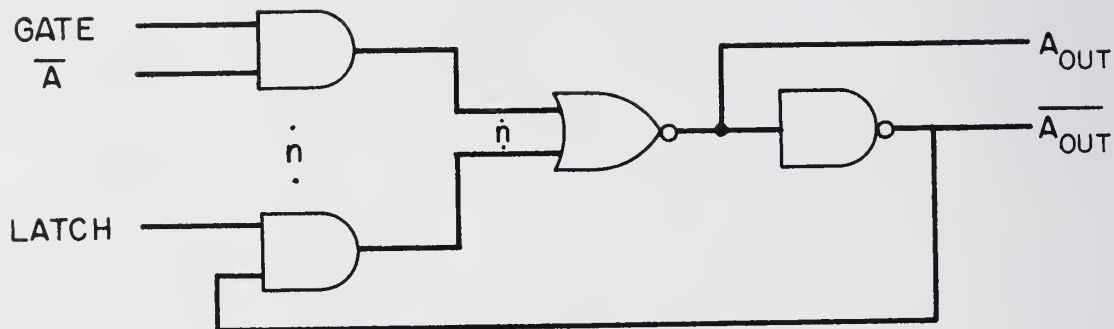
#### 2.1.1.6 Gate Inhibits for the LR and IR

As mentioned previously in Section 2.1.1, the Permuter cards physically contain the LR, AR, IR and DR flip-flop storage. These registers are built up from a "latch" type flip-flop which is shown and described in Figure 2.1.1.6/1. In order to load a new value into the latch flip-flop, the gate must be on and the latch signal must be off.

The gating inputs for each of these 4 registers are organized on a byte basis, i.e. each gating signal is used to activate the 4 gate and 4 latch lines to the appropriate register. In this process it is quite simple to impose gate inhibits on the individual bytes of a register, and this is exactly what was done in the case of the DR and the IR.

The rest of this section will describe the LR gate inhibit logic. The IR logic is quite similar and may be found in the TP Logic Book in the Series-04 drawings. The LR gate which is inhibited is the DR-to-LR gate. The logic for this inhibition is shown in Figure 2.1.1.6/2. DLR/G is the main gating signal from the control logic. It is NAND'ed with DLRi/N ( $i = 1, 2, 3, 4$ ), one of 4 inhibit lines. Thus if  $DLRi/N = "1"$ , the gate signal will be inhibited and the gating and latching signals to the  $i$ th byte of the LR will not be activated. Note that other signals such as the right shift enable, RS/E, the AND enable, AND/E, or others can also set the LR latch signal to 0 and the gate signal to 1.

This facility is used mainly in the merging operations in the memory access sequencing. It is also used in other control sequences where it is desired to replace part of the contents of a register and leave the remaining contents unmodified.



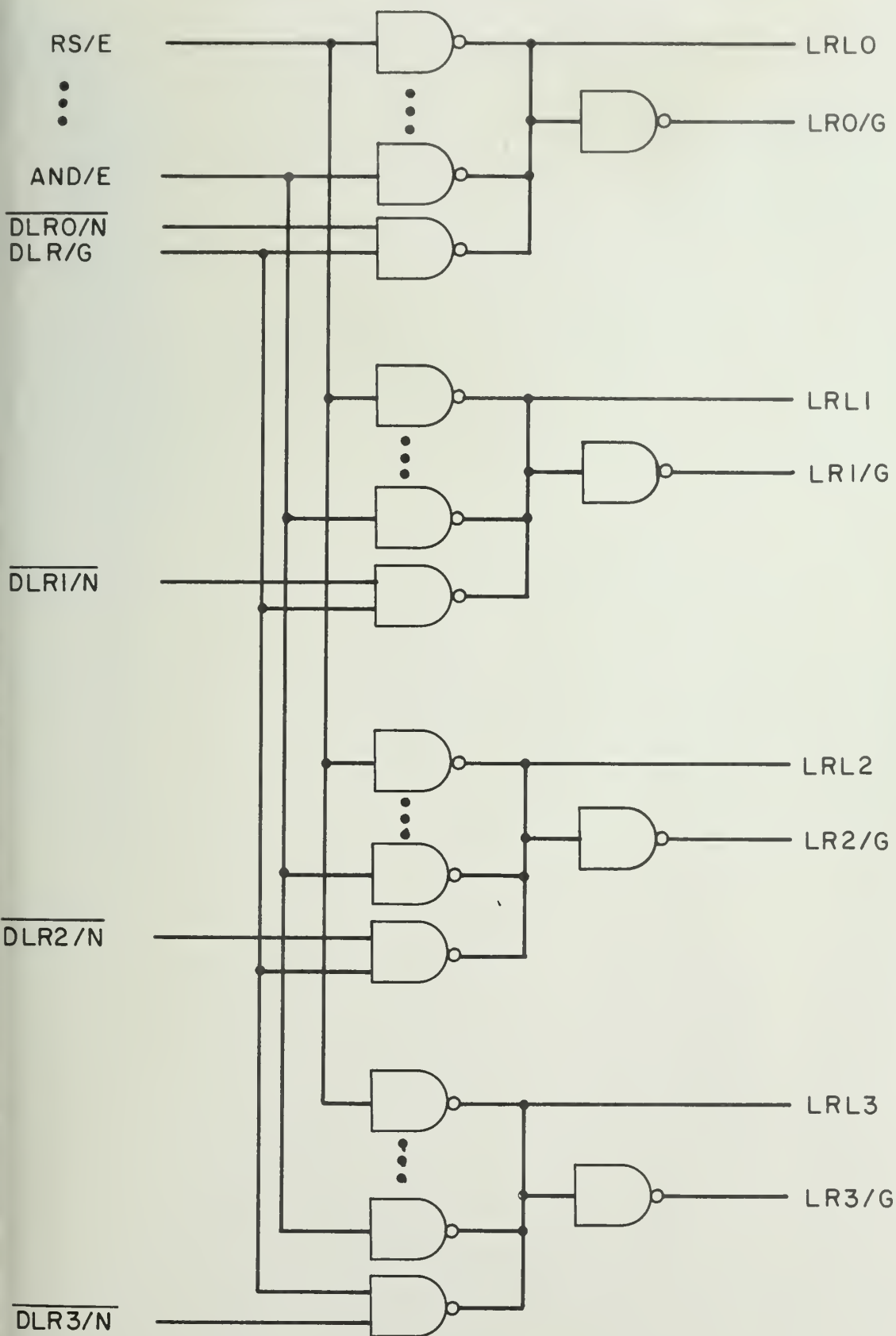
where  $n=0,1,2,\dots$  additional gates

Gate: enables  $\bar{A}$  to be transmitted to the NOR

Latch: when off, allows the output of the gate AND's to determine the state of the flip-flop. When ON maintains state of the flip-flop so that gates will not affect it.

Figure 2.1.1.6/1 - Latch Logic Circuits





$LRLi = \text{LATCH}$   
 $LRi/G = \text{GATE}$

FIGURE 2.1.1.6/2 GATE INHIBIT CIRCUIT



#### 2.1.1.7 Permuter Register Gate Drivers

These gate drivers are used to drive the various flip-flop (latch) register and permuter input gates. The various gating lines can be found on the 237-00 card schematic in the circuit book or in the TP Logic Book. Each -237 card contains one bit position of the permuter as well as the LR, DR, AR, IR registers.

There is only a single input to the LR, AR and IR; all the various gating signals are combined in the control gating logic, and the single gate and latch signal needed is sent to the register on a per byte basis. The data inputs are combined in a dot-or (see, for example, the SHIFT/BOOLEAN logic). Note that the LR and AR are loaded with the complement of the desired output.

The gating signals are formed on a byte basis; there is a group of drivers for each register byte (9 bits).

Provision is made to inhibit bytes on certain transfers, these being: DR to LR; adder to AR, and DB to IR. In this case, the inhibit implies that the byte inhibited is not transferred into the register in question. The original register contents remain unchanged however -- nothing is set to zero. This feature allows bytes to be masked into a common register (c.f. PR transfer logic).



## 2.1.2 Signal Name Lists for the Permuter

### 2.1.2.1 Control Signals

#### Input Select Gates:

ARP/G/	-	AR is the permuter input storage block
BR/S	-	Select BR from BR-IBR
BRSP/G/	-	BR-IBR Storage Block output bus (BRØSi) is the permuter input
CTP/G/	-	Cable terminators is the permuter input
DRP/G/	-	DR is the permuter input
IBR/S	-	Select IBR from BR-IBR Storage Block
ICTP/G	-	Mask ICT into low-order 3 bits of DB
IRP/G/	-	IR is the permuter input
LRP/G/	-	LR is the permuter input
OSP/G/	-	Operand stack output bus (BRØSi) is the permuter input
PRP/G/	-	Pointer register output bus (PRBi) is the permuter input
SCRPG/G/	-	Mask SCR into low-order 5 bits of DB via OSR bus and permuter

#### Constant Generator Enable Signals:

M1/E/	-	Generate -1
M2/E/	-	Generate -2
M4/E/	-	Generate -4
M8/E/	-	Generate -8
M32/E/	-	Generate -32
P1/E	-	Generate +1
P2/E	-	Generate +2
P4/E	-	Generate +4
P8/E	-	Generate +8
P32/E	-	Generate +32

## Register Transfer Gate Signals

ADR/G/	-	Gate AR to DR - direct
BAR/G/	-	Gate DB to AR
BDR/G/	-	Gate DBP to DR
BIR/G/	-	Gate DBP to IR
BLR/G/	-	Gate DB to LR
CDR/E/	-	Gate DB + spares to cable driver
DLR/G/	-	Gate DR to LR - direct
LDR/G/	-	Gate LR to DR - direct

## Register Transfer Byte Inhibit Signals:

ADD0/N/	
ADD1/N/	Inhibit adder output byte i
ADD2/N/	
ADD3/N/	
BIR0/N/	
BIR1/N/	Inhibit byte i on DBP to IR gate
BIR2/N/	
BIR3/N/	
DLR0/N/	
DLR1/N/	Inhibit byte i on DR to LR gate
DLR2/N/	
DLR3/N/	

## Permutation Control Signals:

- BNR/G - Base name register gate. Uses contents of base name register to control permutation of data into or out of (depending on RSD/E) base register storage.
- CRRL/G - Gate "core" to registers (DR, LR, etc.) left justified
- LJI/E - Left justify
- LRS/E - Inhibits normal operation of permutation control logic.

This signal is used by the shift control logic to perform shifts in byte multiples using the PLi and/or IBPLij signals. It must also be used when the PLCi signals are used by the main control logic to generate a permutation. Note! One of the above mentioned signals must be on when LRS/E is on in order for the permuter to work properly. (See Section 2.1.4.3).

- OPJ/E - Used if IBRP/G is also on. Justifies opcode on byte 3 instead of byte 0.
- OSRL/G - Gate OS to registers right justified
- OSRR/G - Gate OS to registers left justified
- OSS/S - Select OS. Turns on when either DBOS/S or DSP/G is 'on'.
- RJI/E - Right justify
- RSD/E - Reverses role of source and destination bits in permutation control





## 2.1.2.2 "Internal" Signals Used by the Permuter

<u>Logic Book Name</u>	<u>Description</u>
ARi	Arithmetic Register - bit i
ARi/G	AR input gating signal, $ARN \rightarrow AR$ - byte i
ARGi/G	Gate signal, $AR \rightarrow DR$ , byte i
ARLi	AR latch, always = $\overline{ARi/G}$ - byte i
ARNi/	Adder output bus = AR input - bit i
BRSP/G	Permuter in gate signal, $BR\emptyset Si \rightarrow FBIi$ , = OSP/G v BRS
CSB	Cell size = byte
CSBH	Cell size = byte or halfword
CSH	Cell size = halfword
CTi	Cable Terminators - bit i
DBi	Distribution Bus, bit i = DBPi
DBDi/G	Gate signal, $DE \rightarrow DR$ - byte i
DBPi	Distribution Bus directly from permuter - bit i
DRi	Distribution Register - bit i
DRLi	DR latch, byte i = 0 whenever something $\rightarrow DR$ - bit i
FBIi	Output from permuter input gate - bit i
IBi/E	Inhibit Enable - byte i
IBPL13	Inhibit byte 1 - permute left 3
IBPL21	Inhibit byte 2 - permute left 1
IBi/	Inhibit signal for byte i - from shift control
ICTi	Instruction byte counter - bit i
IRi	Instruction register - bit i
IRi/G	IR input gating signal $DB \rightarrow IR$ - byte i
IRLi	IR latch, always = $\overline{IRi/G}$ - byte i

Logic Book NameDescription

LRi	Logic Register - bit i
LRDi/G	Gate signal, LR → DR - byte i
LRi/G	LR input gating signal LRN → LR - byte i
LRLi	LR latch, = $\overline{\text{LRi/G}}$ - byte i
LRNi	In-bus to LR - comes from SHIFT/BOOLEAN and PACK-UNPACK, etc. - byte i
OSRi	Bus used to insert ØSTR, SCR and ICT onto permuter output (last 5 bits)
OSTRi	Operand Stack Top register
PLi	Permute left i bytes - output from shift control
PLi/E	Enable permute left i bytes
PRBi	Pointer Register Bus - bit i
SCRi	Stack Control Register - bit i
T2Ri	Temporary Register 2 in Operand Stack - bit

```

EXEC PL1
PL1.SYSPUNCH DD SYSOUT=B
PL1.SYSIN DD *
PER_IN: PROC(ARPG, BNRG,BRGS,      BRSPBG,      CRRLG,      CSB,
            CSHH,      CSH,      CTPG,      DBRJ,      DRPG,      IBPL13,
            IBPL21,      IBRS,      ICTPG,      IRPG,      LJIE,      LJNPE,
            LRPg,      LRSE,      M16E,      M1E,      M2E,      M32E,
            M4E,      M8E,      NRSETG,      NPJE,      OSPG,      OSTPG,
            P16E,      P1E,      P2E,      P32E,      P4E,      P8E,
            PLC1,      PLC2,      PLC3,      PRPG,      RJIE,      RSDE,
            SCRPG,      LRSE1,      IB,      IBC,      PL,      OSSS,
            OSPTG,      OSPFG);
DCL (OSPTG,OSPG) BIT(1);
DCL      LRSE1 BIT(1);
DCL OSSS BIT(1);
DCL( ARPG,BNRG,      BRGS,      BRSPBG,      CRRLG,      CSB,
    CSHH,      CSH,      CTPG,      DBRJ,      DRPG,      IBPL13,
    IBPL21,      IBRS,      ICTPG,      IRPG,      LJIE,      LJNPE,
    LRPg,      LRSE,      M16E,      M1E,      M2E,      M32E,
    M4E,      M8E,      NRSETG,      NPJE,      OSPG,      OSTPG,
    P16E,      P1E,      P2E,      P32E,      P4E,      P8E,
    PLC1,      PLC2,      PLC3,      PRPG,      RJIE,      RSDE,
    SCRPG) BIT(1), (IB, IBC,      PL) (0:3) BIT(1);
DCL( (AR,BROS,CT,DB,DBB,DBP,DR,IR,LR,PRB)(36), (BNR,ICT)(3),
    (ACT,CCT)(4), (SCR,T2R)(5), OSTR(2), BRS(0:7))
    BIT(1) EXTERNAL;
DCL (PLIOE,PLI1E,PLI2E,PLI3E) BIT(1);
DCL FBI(36) BIT(1),IBE(0:2)BIT(1),IBE3(1:9)BIT(1),
    (S,D)(0:1)BIT(1),OSR(1:5) BIT(1),
    (PLOE,PL1E,PL2E,PL3E,BRSPG) BIT(1),
    (I,J) FIXED BIN;
OSPG=OSPTG|OSPG;
/* DETERMINE LENGTH OF SHIFT FROM SOURCE AND DESTI-
NATION POSITIONS, S AND D. IF LRSE IS ON INHIBIT
NORMAL CONTROL.*/

LRSE=M1E|M2E|M4E|M8E|M16E|M32E|LRSE1|PLC1|PLC2|PLC3;
IF LRSE THEN DO;
    PLIOE,PLI1E,PLI2E,PLI3E='0'B;
    GO TO SKIPSD;
END;

S(0),S(1),D(0),D(1)='0'B;

IF BRGS THEN DO;
    S(0)=BRS(1)|BRS(2)|BRS(5)|BRS(6);
    S(1)=BRS(1)|BRS(3)|BRS(5)|BRS(7);
END;

IF DBRJ THEN DO;
    D(0)=CSH;
    D(1)=CSB;
END;

IF OSSS THEN DO;
    S(0)=T2R(4);
    S(1)=T2R(5);
END;

```

```

IF BNRG THEN DO;
    S(1)=BNR(3);
    S(0)=BNR(2)&(¬BNR(3))
        |BNR(3)&(¬BNR(2));
END;

IF CRRLG THEN DO;
    S(0)=AR(34);
    S(1)=AR(35);
END;

IF IBRS THEN DO;
    S(0)=ICT(2);
    S(1)=ICT(3);
    D(1)=OPJE;
END;

IF LJNPE THEN D(1)='0'B;
IF RSDE THEN
    BEGIN;
        DCL TEMP(0:1) BIT(1);
        TEMP=S;
        S=D;
        D=TEMP;
    END;

DO;
    PLIOE= ¬S(0)&¬S(1)&¬D(0)&¬D(1)
        |¬S(0)& S(1)&¬D(0)& D(1)
        | S(0)&¬S(1)& D(0)&¬D(1)
        | S(0)& S(1)& D(0)& D(1);
    PLIIE= ¬S(0)& S(1)&¬D(0)&¬D(1)
        | S(0)&¬S(1)&¬D(0)& D(1)
        | S(0)& S(1)& D(0)&¬D(1)
        |¬S(0)&¬S(1)& D(0)& D(1);
    PLI2E= S(0)&¬S(1)&¬D(0)&¬D(1)
        | S(0)& S(1)&¬D(0)& D(1)
        |¬S(0)&¬S(1)& D(0)&¬D(1)
        |¬S(0)& S(1)& D(0)& D(1);
    PLI3E= S(0)& S(1)&¬D(0)&¬D(1)
        |¬S(0)&¬S(1)&¬D(0)& D(1)
        |¬S(0)& S(1)& D(0)&¬D(1)
        | S(0)&¬S(1)& D(0)& D(1);
END;

SKIPSD: PLOE=PLIOE|PL(0);
        PL1F=PLIIE|PL(1)|IRPL21|PLC1;
        PL2F=PLI2E|PL(2)|PLC2;
        PL3E=PLI3E|PL(3)|IBPL13|PLC3;

/* SELECT PROPER INPUT TO BE GATED*/

FBI='0'B;
BRSPG='0'B;
IF DRPG THEN FBI=DR;

```

```

IF IRPG THEN FBI=IR;
IF LRPG THEN FBI=LR;
IF ARPG THEN FBI=AR;
IF PRPG THEN FBI=PRB;
IF CTPG THEN FBI=CT;
IF OSPTG|OSPEG|KRSBPG THEN DO;
    BRSPG='1'B; FBI=BRBS; END;

```

```

/* CHECK FOR MASKING */

```

```

IF OSTPG | SCRPG THEN DO I=31 TO 35;
    FBI(I)='1'B;
    END;
IF ICTPG THEN DO I=33 TO 35;
    FBI(I)='1'B;
    END;
IF NRSETG THEN DO I=1 TO 4;
    FBI(I+27)=ACT(I);
    FBI(I+31)=CCT(I);
    END;

```

```

/* DETERMINE INPUTS TO INHIBIT ENABLE NETWORK FROM
OPERAND STACK.*/

```

```

OSR='1'B;
IF OSTPG THEN DO;
    OSR(1)=OSTR(1);
    OSR(2)=OSTR(2);
    OSR(3),OSR(4),OSR(5)='0'B;
    END;

```

```

IF SCRPG THEN OSR=SCR;

```

```

/* DETERMINE INPUTS TO INHIBIT ENABLE NETWORK FROM
INSTRUCTION BYTE COUNTER.*/

```

```

IF ICTPG THEN DO;
    OSR(1),OSR(2)='0'B;
    DO I=1 TO 3;
        OSR(I+2)=ICT(I);
    END;
    END;

```

```

/* CALCULATE INHIBITS.*/

```

```

IBE='0'B;
DO I=1 TO 3; IBE3(I)='0'B; END;
DO I=1 TO 5; IBE3(I+3)=~OSR(I); END;

```

```

IF RJIE THEN DO;
    IF CSBH THEN IBE(0)='1'B;
    IF CSH THEN IBE(1)='1'B;
    IF CSB THEN IBE(1),IBE(2)='1'B;
    END;

```

```

IF LJIF THEN DO;
    IF CSRH THEN IBE3='1'B;
    IF CSH THEN IBE(2)='1'B;

```

```

        IF CSB THEN IBE(1),IBE(2)='1'B;
        END;

IF IBPL13 THEN IBE(1)='1'B;
IF IBPL21 THEN IBE(2)='1'B;
IF IB(0)||IBC(0) THEN IBE(0)='1'B;
IF IB(1)||IBC(1) THEN IBE(1)='1'B;
IF IB(2)||IBC(2) THEN IBE(2)='1'B;
IF IB(3)||IBC(3) THEN IBE3='1'B;

/* CHECK FOR INHIBITS FROM CONSTANT GENERATOR.*/

IF M1E THEN IBE3(8)='1'B;
IF M2E THEN IBE3(7)='1'B;
IF M4E THEN IBE3(6)='1'B;
IF M8E THEN IBE3(5)='1'B;
IF M16E THEN IBE3(4)='1'B;
IF M32E THEN IBE3(3)='1'B;

IF P1E THEN FBI(35)='1'B;
IF P2E THEN FBI(34)='1'B;
IF P4E THEN FBI(33)='1'B;
IF P8E THEN FBI(32)='1'B;
IF P16E THEN FBI(31)='1'B;
IF P32E THEN FBI(30)='1'B;
DBP='1'B;

/* SELECT AND PERFORM SHIFTS.*/

IF PLOE THEN DBP=FBI;
IF PL1E THEN DO;
        DO I=1 TO 27;
        DBP(I)=FBI(I+9);
        END;
        DO I=1 TO 9;
        DBP(I+27)=FBI(I);
        END;
    END;
IF PL2E THEN DO I=1 TO 18;
        DBP(I)=FBI(I+18);
        DBP(I+18)=FBI(I);
    END;
IF PL3E THEN DO;
        DO I=1 TO 9;
        DBP(I)=FBI(I+27);
        END;
        DO I=1 TO 27;
        DBP(I+9)=FBI(I);
        END;
    END;

/* CHECK IB INHIBITS.*/

IF IBF(0) THEN DO I=1 TO 9;
        DBP(I)='0'B;

```

```

                END;
IF IBF(1) THEN DO I=1 TO 9;
                DBP(I+9)='0'B;
                END;
IF IBE(2) THEN DO I=1 TO 9;
                DBP(I+18)='0'B;
                END;
DO I=1 TO 9;
    IF IBE3(1) THEN DBP(I+27)='0'B;
    END;
/* MAKE DB OUTPUTS EQUAL TO DBP OUTPUTS.*/

DB,DBB=DBP;
END PER_IN;

```

```

PER_OUT: PROC(ADDE,      ADRG,      ASRWE,      BARG,      BDRG,
                BIRG,      BLRG,      BRGS,      BSR1G,      BSR2G,
                DBPRG,      DBSTG,      DLRG,      EQVE,      FLPRN,
                LDRG,      LSE,      RSE,      RWBGE,      SBRS,
                WRBRE,      WRPRLE,      WRPRVE,      WSBLE,      WSBVE,
                XORE,      XRVE,      ANDE,
                ADDN,      ARG,      ARGG,      ARL,      BIRN,
                DBDG,      DLRN,      DRL,      IRG,      IRL,
                LRDG,      LRG,      LRL,      DBOSG,      CDRE,
                ARF33,      ARF34,      ARF35,      ARAREF);
DCL(ADDE, ADRG,      ANDE,      ASRWE,      BARG,      BDRG,
    BIRG,      BLRG,      BRGS,      BSR1G,      BSR2G,
    DBPRG,      DBSTG,      DLRG,      EQVE,      FLPRN,
    LDRG,      LSE,      RSE,      RWBGE,      SBRS,
    WRBRE,      WRPRLE,      WRPRVE,      WSBLE,      WSBVE,
    XORE,      XRVE,      DBOSG,      CDRE) BIT(1),
    (ADDN,      ARG,      ARGG,      ARL,      BIRN,
    DBDG,      DLRN,      DRL,      IRG,      IRL,
    LRDG,      LRG,      LRL)(0:3) BIT(1) ;
DCL (ARF33,ARF34,ARF35,ARAREF) BIT(1);
DCL ( (AR,ARN,DB,DBB,DBP,DR,IR,LR,LRN,SR,T)(36),OSTR(2),
    SCR(5), (ASRWE1,ASRWE2,RBRS)(7), BRS(0:7),
    PRS(1:14), (PSNWS1,PSNWS2)(0:15), OSS(0:31),
    BR(0:7,36), SNR(0:14,4), PR(0:14,36),
    PRSNR(0:15,18), OS(0:31,9), ASRG(7,18)) BIT(1)
    EXTERNAL;
DCL(ARB,LRB,DRB) (36) BIT(1) EXTERNAL;
DCL CD(36) BIT(1) EXTERNAL;
    IF CDRE THEN CD=DB;
    /* TRANSFER T TO LRN      DRAWING 07-1 */
    XRVE=XORE|EQVE;
    IF XRVE THEN DO;
        LRN=-T;
        DO I=9 TO 36 BY 9;
            IF DB(I)=DR(I) THEN LRN(I)='1'B;
        END;
    END;

/* SEND OUTPUT TO PROPER REGISTERS.*/
    DRL='1'B;
    DBDG,ARGG,LRDG='0'B;

    IF BDRG|ADRG|LDRG THEN DRL='0'B;

    IF BDRG THEN DO;
        DBDG='1'B;
        DR=DBP;
    END;

    IF ADRG THEN DO;
        ARGG='1'B;
        DR=AR;
    END;

    IF LDRG THEN DO;
        LRDG='1'B;

```



```

        DR=LR;
        END;
    IF BLRG THEN LRN(*)=DB(*)|LRN(*) ;
    IF DLRG THEN LRN(*)=DR(*)|LRN(*) ;
    /* CALCULATE LR GATE AND LATCH SIGNALS  DRAWING 04-4 */
    DO I=0 TO 3;
    LRG(I)=RSE|LSE|~DLRN(I)&DLRG|BLRG|ANDE|XRVE;
        END;
    LRL=~LRG;

    DO I=0 TO 3;
        IF LRG(I) THEN DO J=I*9+1 TO I*9+9;
            LR(J)=LRN(J);
        FND;
    END;

    IF BIRG THEN DO I=0 TO 3;
        IF ~BIRN(I) THEN DO;
            IRL(I)='0'B;
            IRG(I)='1'B;
            DO J=I*9+1 TO I*9+9;
                IR(J)=DBP(J);
            END;
        END;
    END;

    IF BARG THEN ARN=ARN|DB;
    DO I=0 TO 3;
        ARG(I)=BARG|ADDE&~ADDN(I);
    END;
    ARG(0)=ARG(0)|RWBGE;
    ARL=~ARG;
    DO I=0 TO 3;
        IF ARG(I) THEN DO J=I*9+1 TO I*9+9;
            AR(J)=ARN(J);
        END;
    END;

    IF ARARFG THEN DO;
        ARF33=AR(33);
        ARF34=AR(34);
        ARF35=AR(35);
    END;

    /* CHECK FOR INITIALIZATION OF STACK */
    /* DBSTG WILL BE ON IF THE OSTR AND SCR ARE TO BE RELOADED
    WITH A NEW OPERAND STACK POINTER ADDRESS */

    IF DBSTG THEN DO;
        DO I=1 TO 2;
            OSTR(I)=DB(I+30);
        END;

        DO I=1 TO 5;
            SCR(I)=DB(I+30);
        FND;
    END;

```

```

/* LOAD OS FROM DB */

IF DBOSG THEN DO I=0 TO 31;
    IF OSS(I) THEN DO;
        L=MOD(I,4);
        DO J=1 TO 9;
            OS(I,J)=DB(L*9+J);
        END;
    END;
END;

/* WRITE ON SELECT POINTER OR SBR IF DESIRED */
IF DBPRG THEN DO;
    IF WRPRLE THEN DO I=0 TO 14;
        IF PRS(I) THEN DO;
            IF FLPRN THEN DO;
                DO J=1 TO 8;
                    PR(I,J)=DBP(J);
                END;
                DO J=10 TO 17;
                    PR(I,J)=DBP(J);
                END;
            END;
        ELSE DO J=1 TO 18;
            PR(I,J)=DBP(J);
        END;
    END;
END;

IF WRPRVE THEN DO I=0 TO 14;
    IF PRS(I) THEN DO;
        IF FLPRN THEN DO;
            DO J=19 TO 26;
                PR(I,J)=DBP(J);
            END;
            DO J=28 TO 35;
                PR(I,J)=DBP(J);
            END;
        ELSE DO J=19 TO 36;
            PR(I,J)=DBP(J);
        END;
    END;
END;

IF WSBLF&SBR5 THEN
    DO I=1 TO 18;
        SBR(I)=DBP(I);
    END;

IF WSBVF & SBR5 THEN
    DO I=19 TO 36;
        SBR(I)=DBP(I);
    END;

/* INTERRUPT RECOVERY FOR SHADOW NAME REGISTER */
IF RSRIG THEN DO I=0 TO 2;
    DO J=1 TO 4;
        SNR(2*I+8,J)=DB(I*9+J);
    END;
END;

```

```

        SNR(2*I+9,J)=DB(I*9+J+4);
    END;
    DO J=1 TO 4;
        SNR(14,J)=DB(27+J);
    END;
END;
IF BSR2G THEN DO I=0 TO 3;
    DO J=1 TO 4;
        SNR(2*I,J)=DB(I*9+J);
        SNR(2*I+1,J)=DB(I*9+J+4);
    END;
END;
END;

/* CHECK FOR WRITING INTO THE POINTER REGISTER SEGMENT NAME
   REGISTER PRSNR */
/* DEFINE PRSNR(0:15,18) AS THE INTERNAL REGISTERS ON
   DRAWINGS 22-4 TO 22-19 IN THE LOGIC BOOK */
DO I=0 TO 15;
    IF PSNWS1(I) THEN DO J=1 TO 8;
        PRSNR(I,J)=DBB(J);
    END;
END;
DO I=0 TO 15;
    IF PSNWS2(I) THEN DO J=10 TO 17;
        PRSNR(I,J)=DBB(J);
    END;
END;
END;

/* ASSOCIATIVE REGISTER WRITE SIGNAL DRIVERS */
IF ASRWE THEN DO I=1 TO 7;
    IF RBR5(I) THEN DO;
        ASRWE1(I),ASRWE2(I)='1'B;
        DO J=1 TO 17 WHILE (J≠9);
            ASRG(I,J)=DBB(J);
        END;
    END;
END;
ELSE ASRWE1,ASRWE2='0'B;

/* WRITE INTO BASE REGISTER */
IF BRGS&WRBRE THEN DO I=0 TO 7;
    IF BRS(I) THEN BR(I,*)=DBP(*);
END;
LRB=LR;
ARB=AR;
DRB=DR;
END PER_OUT;
/*

```



## 2.1.4 Permuter - Logic Description

### 2.1.4.1 The Basic Permuter Card

The heart of the Permuter logic is the -237 card shown in Figure 2.1.4.1. This card consists of four latch flip-flops plus two NOR's, each driven by several AND circuits.

The flip-flops are designated LR, AR, IR, and DR. The first two are independent of any direct action taken by the rest of the card circuitry (i.e. there are no direct connections to the LR and AR flip-flops from the other circuits).

In the LR flip-flop note that the complement of the input ( $\overline{\text{LRNi}}$ ) is stored when the gate signal,  $\text{LRi/G}$ , is logically "1" (+6v) and the latch signal,  $\text{LRLi}$ , is logically "0" (0v). After the data has been stored in the flip-flop, the latch signal must be "1" and the gate must be "0". In order to maintain the new state reliably the latch must become "1" before the gate becomes "0". Otherwise the input signal may be cut off before the new state is "locked in".

The AR flip-flop works in exactly the same manner as the LR flip-flop. The IR flip-flop is also similar except that the data input is from the second of the 2 NOR's on the -237 card. Since this input ( $\text{DBPi}$ ) is not in complemented form the output definitions are reversed. In effect, by connecting the input of the IR flip-flop internally to the card we are restricting its input so that it can only be loaded directly from the Distribution Bus (DBP).

The DR flip-flop is like the IR flip-flop, in that its output definitions are reversed. However, in addition it has three inputs instead of the usual one. One of the inputs is from the DBP as in the IR. The other two are from the LR and AR. Each input has its own separate

gate. Note that in order to store data from one of the three inputs the latching signal (DRLi) must be logically "0" and one of the three gating signals must be "1". If more than one gate is on, the result in the DR will be the OR function of the data on the lines with the gate signals on. Once again, when the data is stored the latch signal should be turned on before the gate signal is turned off.

The first NOR on the -237 card is what has been called the Permuter input driver (see Section 2.1.1.1 and Figure 2.1.1.5/1). All of the possible inputs to the permuter are AND'ed with their respective gating signals and then NOR'ed together to produce the  $\overline{\text{FBI}}_i$  signal. This signal is, in effect, the complement of the desired input to the Permuter. As mentioned in Section 2.1.1.5, various signals from other cards may be dot-OR'ed with this line to cause masking on certain parts of the input.

The final NOR circuit on the -237 card controls the output from the permuter. The first four inputs to this NOR are gated by the Permutation Control signals, PLO/E, PL1/E, PL2/E and PL3/E. At most only one of these will be on at any given time. The data which is gated by these signals comes from an  $\overline{\text{FBI}}$  line on one of four of the Permuter cards. PLO/E gates in the  $\overline{\text{FBI}}$  signal from the same card. PL1/E gates in the signal from 9 bit positions to the right of the present positions (i.e.  $i + 9$ ), PL2/E gates the  $\overline{\text{FBI}}$  from 18 positions to the right, etc.

The last of the inputs to the final NOR is the inhibit signal which causes the DBP to become "0" if it is turned on. It should be noted again that DBP and DB are logically equivalent (see Section 2.1.1.4).







#### 2.1.4.2 Inhibit Generator

The Inhibit Generator for the Permuter is contained in Drawing 04-5 in the TP Logic Book. It is used to determine the inhibit signals sent to pin V on the -237 cards. It should be noted that in bytes 0, 1 and 2 of the Permuter output, the whole byte is inhibited by the same signal, i.e.  $\overline{IB0/E}$ ,  $\overline{IB1/E}$ , or  $\overline{IB2/E}$ , whereas in byte 3 of the Permuter output each of the nine bits can be inhibited individually (see Section 2.1.1.5).

As shown in Figure 2.1.4.2 the inhibit signals can be used for right and left justification of data and for shifting data on a byte basis. A series of five NAND's decode the right and left justification control signals and the cell size signals to determine which inhibit signals should be turned on. The outputs of these NAND's become "0" if the inhibit signal to which they eventually lead must be turned on. In addition to these decoders there are six signals which come from the Boolean/Shift circuits:  $\overline{IB0}$ ,  $\overline{IB1}$ ,  $\overline{IB2}$ ,  $\overline{IB3}$ ,  $\overline{IBPL21}$  and  $\overline{IBPL13}$ . These signals are produced by the Boolean/Shift logic and therefore their generation will be explained in detail in Section 2.8.

The first 4 of these 6 signals are obvious in their meaning and are fed directly to the inhibit signal drivers. The last two actually serve two purposes.  $\overline{IBPL21}$  indicates that byte two should be inhibited and also that there should be a left permutation of 1 byte. The permutation will be discussed later. For now it should be noted that this signal is treated the same as  $\overline{IB2}$ . The  $\overline{IBPL13}$  is the same type of signal and indicates that byte number 1 is to be inhibited and that a left permutation of 3 bytes is to take place. This signal is treated similarly to  $\overline{IB1}$ .

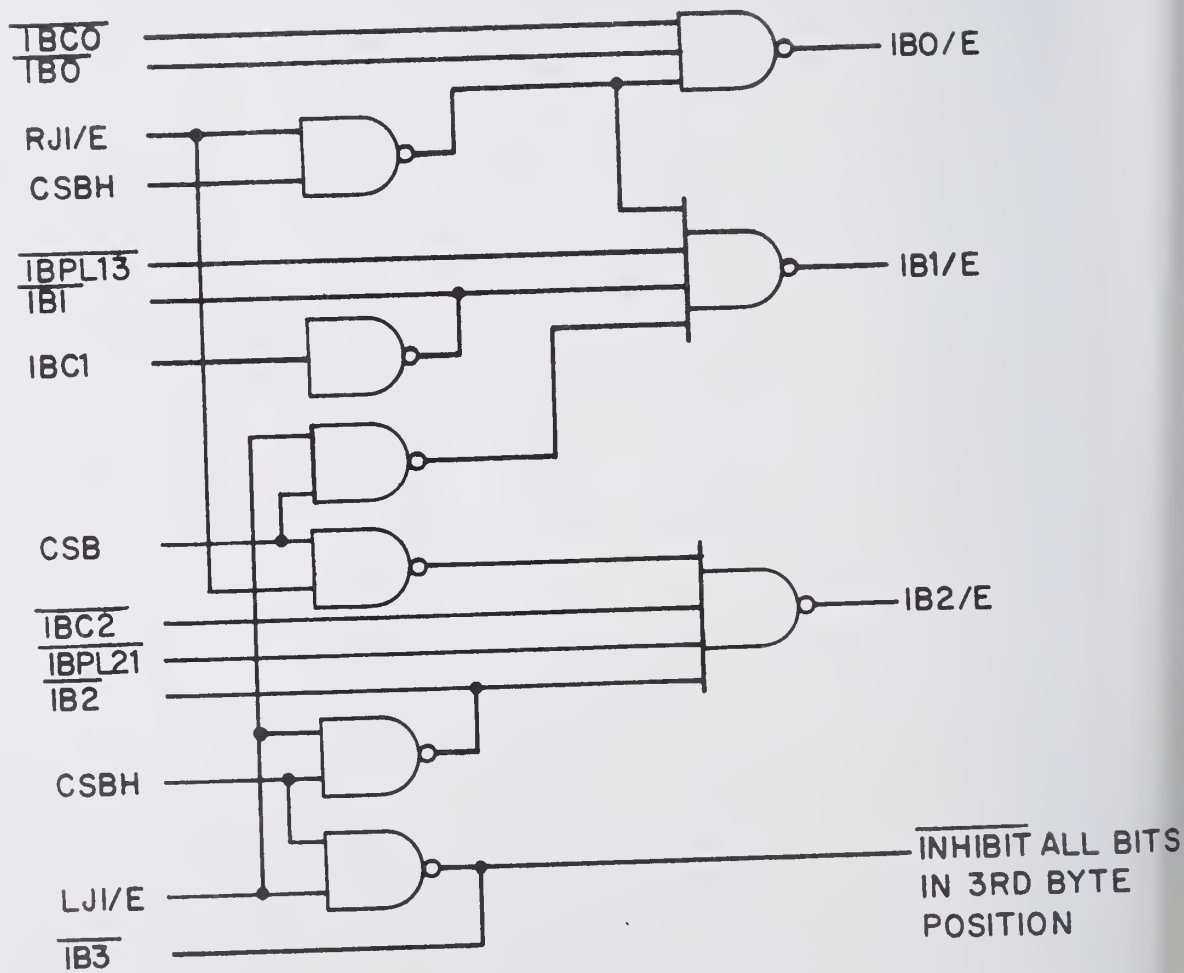


FIGURE 2.1.4.2 INHIBIT GENERATION LOGIC

There are 3 NAND circuits which act as drivers for IBO/E, IB1/E and IB2/E. If one of these is to be turned on, one of the inputs to its respective driver will turn to "0". In the case of byte number 3, however, there are actually nine separate drivers, one for each bit in the byte. If the whole byte is to be inhibited one input in each bit driver is set to "0". Otherwise the input rests at "1" and each driver output may thus be controlled individually by the other driver inputs. These individual inhibit signals may be used as explained in Section 2.1.1.5



### 2.1.4.3 Permutation Control Logic

The permutation control signals, PLO/E, PL1/E, PL2/E, and PL3/E, are obtained either directly from a set of control signals or indirectly by decoding the contents of the 2 bit source and destination busses. The set of control signals originates from the shift control logic in drawing 17-1 of the TP Logic Book or from the main control logic drivers. The permutation control logic itself is on drawing 04-7.

If the set of control signals is used to generate the permutation control signals, the decoding network must be disabled. This job is performed by the LRS/E<sup>1</sup> signal which is inverted and used to drive the enabling input of the decoding logic.

It should be noted that when PLC1, PLC2, or PLC3 is used to control the permutation, the Main Control logic is responsible for turning LRS/E on. This is done at the final driver stage in the control logic just before the signals are sent to the Permuter itself.

It is also important to notice that if LRS/E is turned on and none of the permutation control signals are activated, none of the input data will get through the Permuter. Instead, the output will be all 1's since all of the permutation signals (even PLO/E) will be off. It is exactly this situation that is taken advantage of in the generation of negative constants (see section 2.1.15). The above mentioned logic in the Main Control section is also responsible in this instance to make sure that LRS/E is turned on if M1/E, M2/E, M4/E, M8/E, M16/E, or M32/E is turned on.

---

<sup>1</sup>The name of this signal derives from the fact that when originally conceived, it was used to allow the Left/Right Shift logic to control the permutation.

Whenever the source and destination busses are used to determine the permutation, the set of control signals must be off and LRS/E must be off. This latter condition enables the 236-15, 4 bit decoding diode matrix board (see figure 2.1.4.3/1) to drive 16, 228-05 circuits. These circuits in turn are dot-or'ed and fed to the 4 NAND circuits driving the PLi/E signals according to the equations given in section 2.1.1.3.

The inputs to the decoding circuit consist of the true and complement signals from two, 2-bit busses. These two busses are loaded from the source and destination busses by the Reverse Source and Destination enable signal,  $\overline{RSD/E}$ . When  $\overline{RSD/E} = 1$ , the source bus is gated to the first 2 bit input bus to the decoder (INA1 and INA2 in figure 2.1.4.3/1) and the destination bus is gated to the second input bus (INB1 and INB2 in figure 2.1.4.3/1). When  $\overline{RSD/E} = 0$ , the gating is reversed and the source bus goes to INB and the destination bus goes to INA.

It should be noted that  $\overline{RSD/E}$  is normally automatically activated by special logic in the Main control logic final driver stage where it is turned on by any signal which indicates that there will be a write in one of the storage areas.

The source and destination busses themselves are generated by dot-or'ing various types of information onto them depending on the type of registers involved in the transfer and the type of data justification desired. Note that in drawing 04-7, the source and destination busses are in complemented form as they go into the  $\overline{RSD/E}$  and  $\overline{RSD/E}$  gates.

The key point to remember when trying to understand the operation of the logic attached to these busses is that the normal flow of data is considered to be from storage areas (ie.memory or the storage blocks) to the "working" registers (ie.AR, DR, IR, LR). Thus

when expressing the data justification used by a given storage block, the source bus is always used, even if the data is being loaded into the storage block instead of being read out of it. In the case of loading a storage block the  $\overline{\text{RSD/E}}$  signal will be set to 0 by the control logic and the source bus will then act as the destination input to the decoding logic.

With this in mind we can discuss the various inputs to the source and destination busses.

The DBRJ signal is used when the data is to be right justified in the working registers. In this case, if the cell is a halfword, the "destination" cell will be on the 10 byte boundary and therefore  $\overline{\text{D1}}$  must be forced to '0'. If the cell size is a byte the destination address is 01 and  $\overline{\text{D2}}$  must be set to '0'. Note that for word and doubleword cell sizes no action need be taken since the data will begin at the 00 byte address. In doubleword cells this is due to the fact that the doubleword cell is treated as two word size cells in the Permuter.

When OSS/S is turned on, data is either coming from the OS or going to the OS via the Permuter. The address within the OS, of the cell being accessed is contained in the 5 bit Operand Stack Temporary Register, T2R. In particular, the last two bits of the register, T2R4 and T2R5, indicate the byte boundary address of the cell. Thus, considering the "normal" flow of data through the Permuter, the two source lines must be set equal to the last two T2R bits, ie.  $\overline{\text{S1}} = \overline{\text{T2R4}}$  and  $\overline{\text{S2}} = \overline{\text{T2R5}}$ .

When BRG/S is on, the base register storage block is being accessed. As is explained in section 2.5.1.1, the storage for the base registers is rather mixed up since each register is only 3 bytes long while the storage blocks are based on 8 byte double words. This causes the beginning byte address to vary according to the base register number. Referring to figure 2.5.1.1, it can be seen that



the equations for the two source bits can be written as:

$$S1 = BR1/S \vee BR2/S \vee BR5/S \vee BR6/S$$

$$S2 = BR1/S \vee BR3/S \vee BR5/S \vee BR7/S$$

Two NAND's utilizing inverted inputs are used to produce these signals which are then gated to the  $\overline{S1}$  and  $\overline{S2}$  lines by the BRG/S signal.

The CRRL/G signal is used when accesses are being made to core. In this case the last two bits of the core address determine the source bit settings. These two bits are stored in ARF3<sup>4</sup> and ARF3<sup>5</sup> which are part of a 3 bit register specifically used for storing the low order 3 bits of the core address.

The IBR/S signal indicates that the IBR part of the BR-IBR storage block is to be accessed. The initial address of the word cell to be accessed is determined by the ICT. Thus ICT2 and ICT3 are gated to the  $\overline{S1}$  and  $\overline{S2}$  lines respectively. In addition if both IBR/S and OPJ/E are on the  $\overline{D2}$  line will be activated. This option is used when it is desired to justify the data from the IBR on the 01 byte position of the destination.



## 2.2 Nine Bit/Byte Buffer Storage Blocks

Nine bit/byte storage blocks are used by the Taxicrinic Processor to buffer data for which fast access is required. The basic storage element is the Smith flip-flop<sup>1</sup>, eight of which are contained on one -204 card. By driving nine of these -204 cards with two -205 cards, a storage block containing eight bytes of nine bits each (8 data bits and 1 flag bit), complete with gates and drivers, is formed. These blocks are then used in groups to build up the various data registers in the TP.

Storage blocks are used to form 25 registers in the TP in addition to the fast storage in the Operand Stack. Four blocks are required to form the 32 bytes of the Operand Stack maintained in the TP. Eight blocks form the 15 pointer registers (4 bytes each) and the Spare Buffer Register (4 bytes). And finally 4 blocks are used to form the 8 Base Registers (3 bytes each) and the Instruction Buffer Register (8 bytes).

---

<sup>1</sup>Designed by Professor K. C. Smith



### 2.2.1 Buffer Storage - Functional Description

The organization of the Buffer Storage Blocks is shown in Drawing 01-0 in the TP Logic Book. Each block is made up of two -205 cards which provide the gating and driving logic, and nine -204 cards which provide the actual storage. Each -204 card supplies 1 bit position for each of 8 bytes.

This section defines the data and control signal inputs to a storage block. Operation of the storage block is then described.

The two -205 cards together supply 8 Byte Select Drivers (BSD), 2 Input Gate Drivers (IGD), 2 Output Gate Drivers (OGD) and 6 spare NAND circuits. All inputs to all drivers rest in the logical "1" state.

The BSD's set the proper voltage level on the emitters of the flip-flops in the -204 cards. Each BSD drives one flip-flop on each -204 card, these flip-flops then represent the byte designated by their corresponding BSD. As shown in Figure 2.2.1/1 there are 3 output levels, which determine the "sensitivity" of the 9 driven flip-flops. A "high" output level (+3.5 volts) enables the flip-flop to receive data or to be written into; the "intermediate" level (+1 volt) is the "holding" level at which the flip-flop can neither be written into or read out of. The "low" output level of the BSD (-2.5 volts) causes the flip-flops to be read out via the Output Gates.

The signal levels on the two input control lines to the BSD uniquely determine the state of the output line. One input,

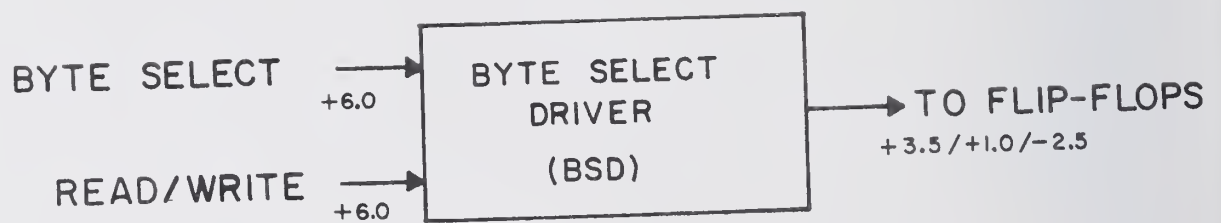


FIGURE 2.2.1/1 BYTE SELECT DRIVER

the Byte Select line, overrides the effects of the second, the Read/Write line. When the Byte Select line is "1", the BSD output is at the intermediate level (+1 volt). If the Byte Select line is "0" the output will be determined by the Read/Write input signal. When this is "1" the output is at -2.5 volts, the "read out" level. A "0" Read/Write signal sets the BSD output to +3.5 volts, or the "write in" state, again only when the Byte Select signal is "0".

As shown in the TP Logic Book Drawing 01-0 the Byte Select lines come from input pins D,H,L, and P on the two -205 cards. The Read/Write input lines come from pins B,E,J, and M.

The Input Gate Drivers, as shown in Figure 2.2.1/2, gate in the data line to the 2 input gates of the flip-flops. The inputs to the IGD's are normally resting at logical "1" (+6 volts) causing the outputs to normally rest at "0" (0 volts). The drivers are operated together in the normal method of operation: for a "gate both in" effect, both inputs to the flip-flop are activated with complementary signals. If only the "gate true in" signal on the IGD is activated, only the "true" input to the flip-flops will be activated and the result in the flip-flop will be an OR of the old and new data. It must be remembered that in order for a "write in" to occur, the BSD must also be in the "write in" state.

On the -205 cards, the IGD's are simply 2 input NAND's which drive NAND circuits on the -204 cards. Each IGD drives 9 NAND circuits, (one byte) one from each -204 card in the block. These NANDS then drive the inputs to the flip-flops. As shown in

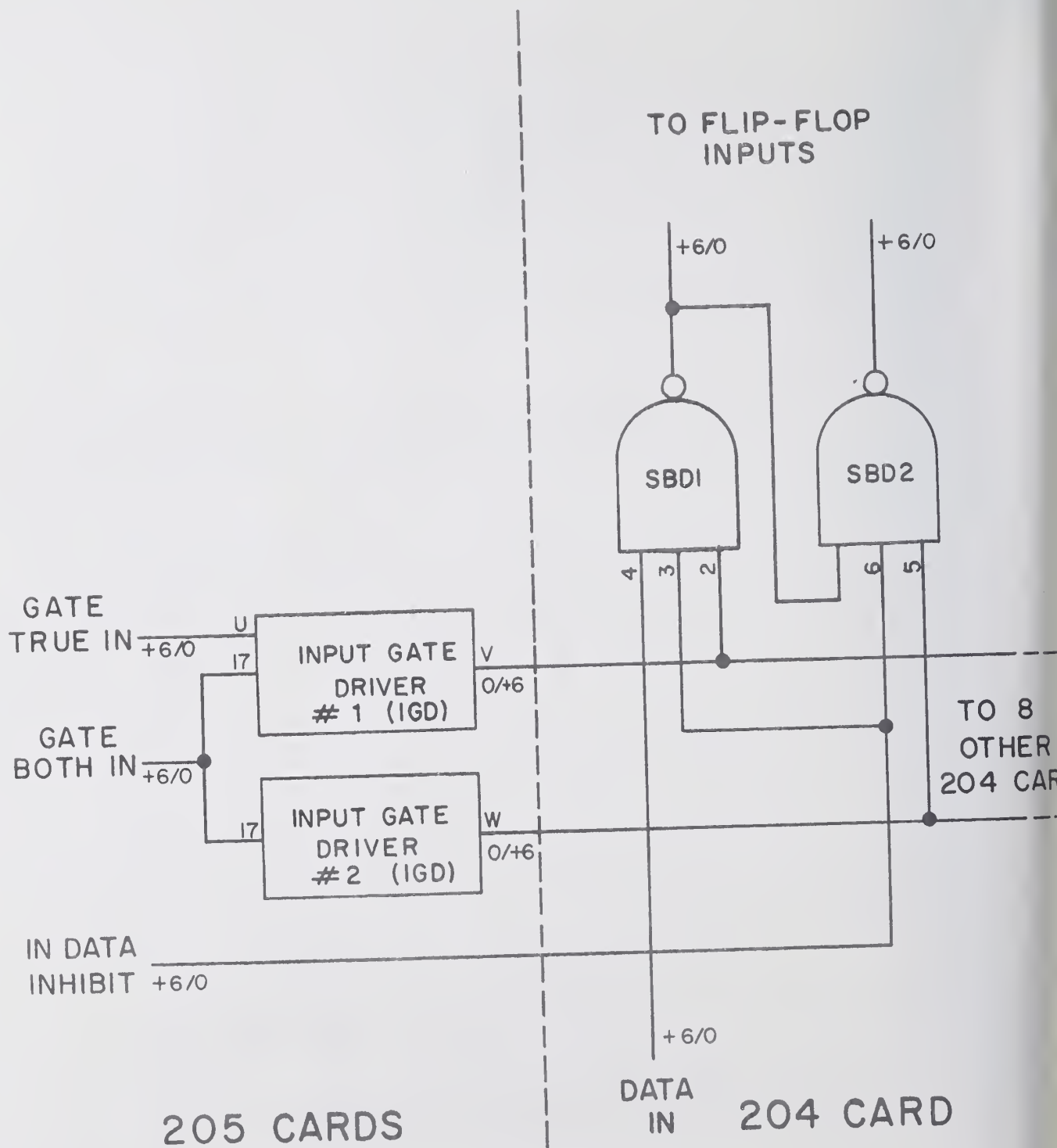


FIGURE 2.2.1/2 INPUT GATE DRIVER

the TP Logic Book Drawing 01-1, the "gate true in" signal to the IGD comes from pin U of the first -205 card. The "gate both in" signal comes from pin 17 of the first -205 card and is tied, internally to the card, to both IGD's.

The In Data Inhibit signal shown in Figure 2.2.1/2 is used in some of the storage blocks (i.e. the Operand Stack) to inhibit the loading of certain bit positions (notably the flags). If the inhibit signal is set to the logical "0" state, the data will not be transmitted to the flip-flops even though the in gates and the select signals are on.

The Output Gate Drivers on the -205 card drive the output gates on the -204 cards. They are shown in Figure 2.2.1/3. The voltage level outputs from the driver are +0.7 and -2.9 volts. Inputs to the driver rest at logical "1", the outputs at logical "0". One OGD is used to drive the true gate from the flip-flop; the other, the false gate. The drivers can be operated either simultaneously for a double-gate effect, or singly, for a gate true (false) out effect. If they are only operated singly, the outputs of the output gates (not the gate drivers) may be tied together (dot OR) to yield a true or complement output from the registers onto a single output line. (This has been done in the Operand Stack buffer storage).

The gating signals into the OGD must be operated when the BSD is set for the "read out" state. Otherwise the outputs from the output gates will both be "0" (0 volts).

The OGD's on the -205 cards consist of a 2 input NAND followed by a driver circuit. There is one OGD on each -205 card. For simultaneous operation, two of the input pins, one from each OGD, are tied together. Pin 20 on the second -205 card connects to the "gate true out".

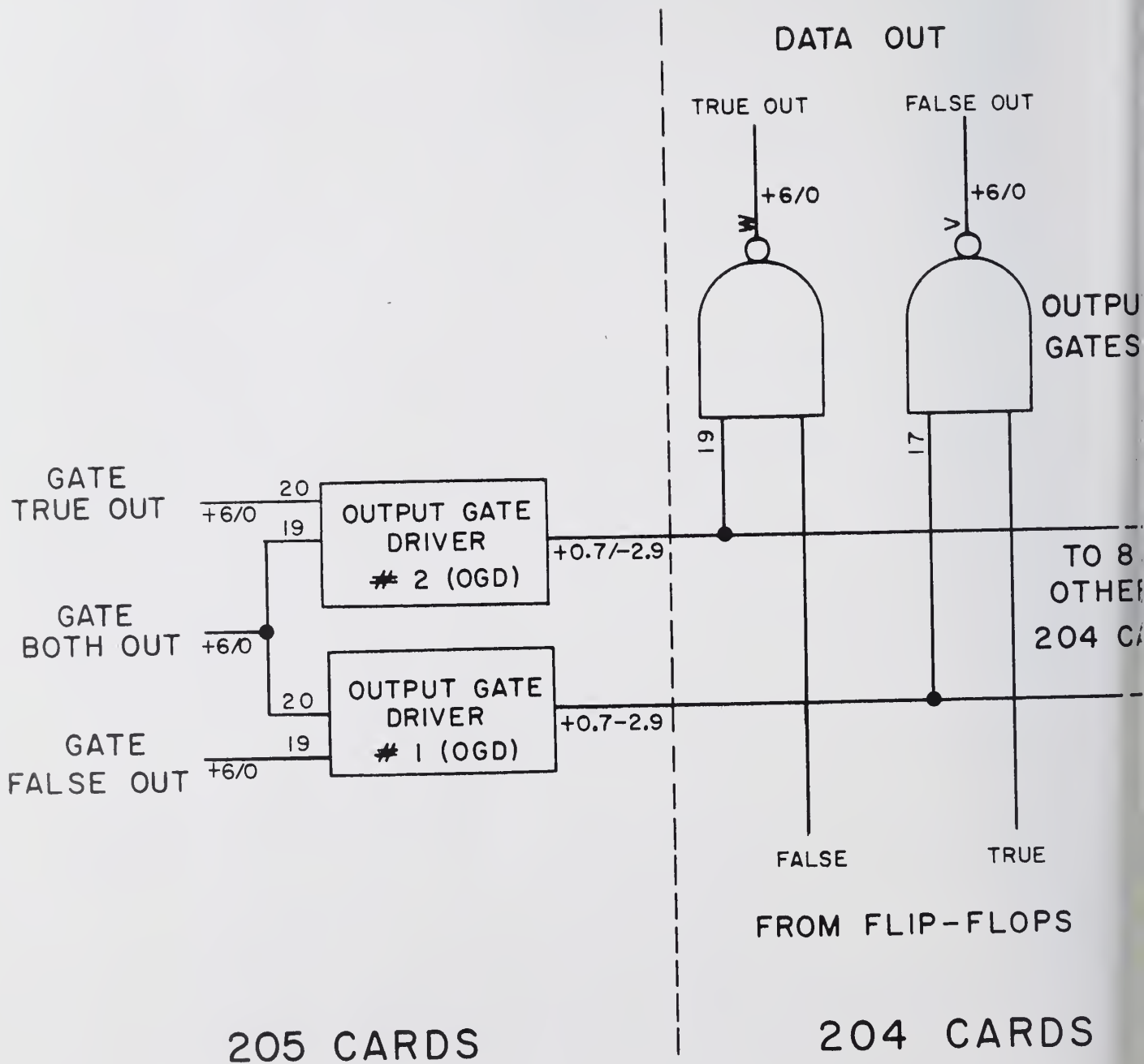


FIGURE 2.2.1/3 OUTPUT GATE DRIVERS



## 2.2.2 Signal Name List for Buffer Storage - Control Signals

Actual Names Used\*

Buffer Storage Name	Base Reg.	Inst. Buf. Reg.	Pointer Reg.	Spare Buffer Register	Operand Stack
Byte Select	$\overline{\text{BRI/S}}$	$\overline{\text{IBRI/S}}$	$\overline{\text{PRi/S}}$	$\overline{\text{SBR/S}}$	$\overline{\text{OSi/S}}$
Read/Write	$\overline{\text{WRBR/E}}$	$\overline{\text{WRIB/E}}$	$\overline{\text{WPRL/E}}$ $\overline{\text{WPRV/E}}$	$\overline{\text{WSBL/E}}$ $\overline{\text{WSBV/E}}$	$\overline{\text{DBOS/G}}$
Gate Both In	$\overleftarrow{\text{DBBRS/G}}$		$\overleftarrow{\text{DBPRS/G}}$		$\overline{\text{DBOS/G}}$
Data In	$\overleftarrow{\text{DBPi}}$				
Gate True Out	$\overleftarrow{\text{BRSBP/G}}$				$\overline{\text{OSPF/G}}$ **
Gate False Out			$\overline{\text{PR7G/S}}$ $\overline{\text{PR8G/S}}$	$\overline{\text{PR8G/S}}$	$\overline{\text{OSP T/G}}$ **
Data Out	$\overleftarrow{\text{BROSi}}$		$\overleftarrow{\text{PRBi}}$		$\overline{\text{BROSi}}$
In Data Inhibit					$\overline{\text{FLOS/N}}$

\*Gate True In and Gate Both Out are provided in the storage blocks but not used.

\*\*Since the outputs from the Operand Stack Storage Blocks are inverted, the Output Gates are the reverse of what would be expected.



#### 2.2.4 Buffer Storage - Logical Description<sup>1</sup>

The purpose of this section is to describe the operation of the buffer storage from a detailed logical and electrical point of view. The description repeats some of the information in Section 2.2.1 but extends to a much more detailed level.

A buffer storage block generally consists of 9, 20<sup>4</sup> storage boards and 2, 20<sup>5</sup> driver boards. The 20<sup>4</sup> board as shown in Figure 2.2.4/1 contains eight flip-flops and some associated circuitry. These eight flip-flops are internally wired to form a "vertical" stack, and in normal operation the nine 20<sup>4</sup> boards are wired side by side (horizontally) to form an 8 x 9 or 72 bit array. The two 20<sup>5</sup> boards are needed to drive the 72 bit array.

During a read or write operation the contents of one of the eight flip-flops will account for one and only one bit of the output register. The nine 20<sup>4</sup> boards together will account for nine bits, i.e. a byte. During the operation of this storage block one byte will be operated on at a time. A typical operation might be sensing the contents of the third flip-flop in each of the nine boards.

In order to be able to work on nine flip-flops at a time, some sort of gating signal must be applied. This is the function of the 20<sup>5</sup> driver boards. Each of these boards contains four circuits which will gate information into and out of the flip-flops. Thus, for a vertical stack of eight flip-flops two such boards are necessary. The remaining circuits on these 20<sup>5</sup> boards are used for controlling the input and output buffers of the entire array.

The key to the operation of this memory is in the type of gating signal supplied by the 20<sup>5</sup> boards. This signal has three distinct levels, and each of these levels puts the flip-flops into a different mode of operation. The different voltages corresponding to these levels are approximately +3.5, +1.0, and -3.0.

<sup>1</sup>The description in this section is the work of H. Magusky.

<sup>2</sup>The design of this board is an outgrowth of the flow gating memory described by Guckel, Kunihiro, and Crow in DCL Report No. 106, March 1961.

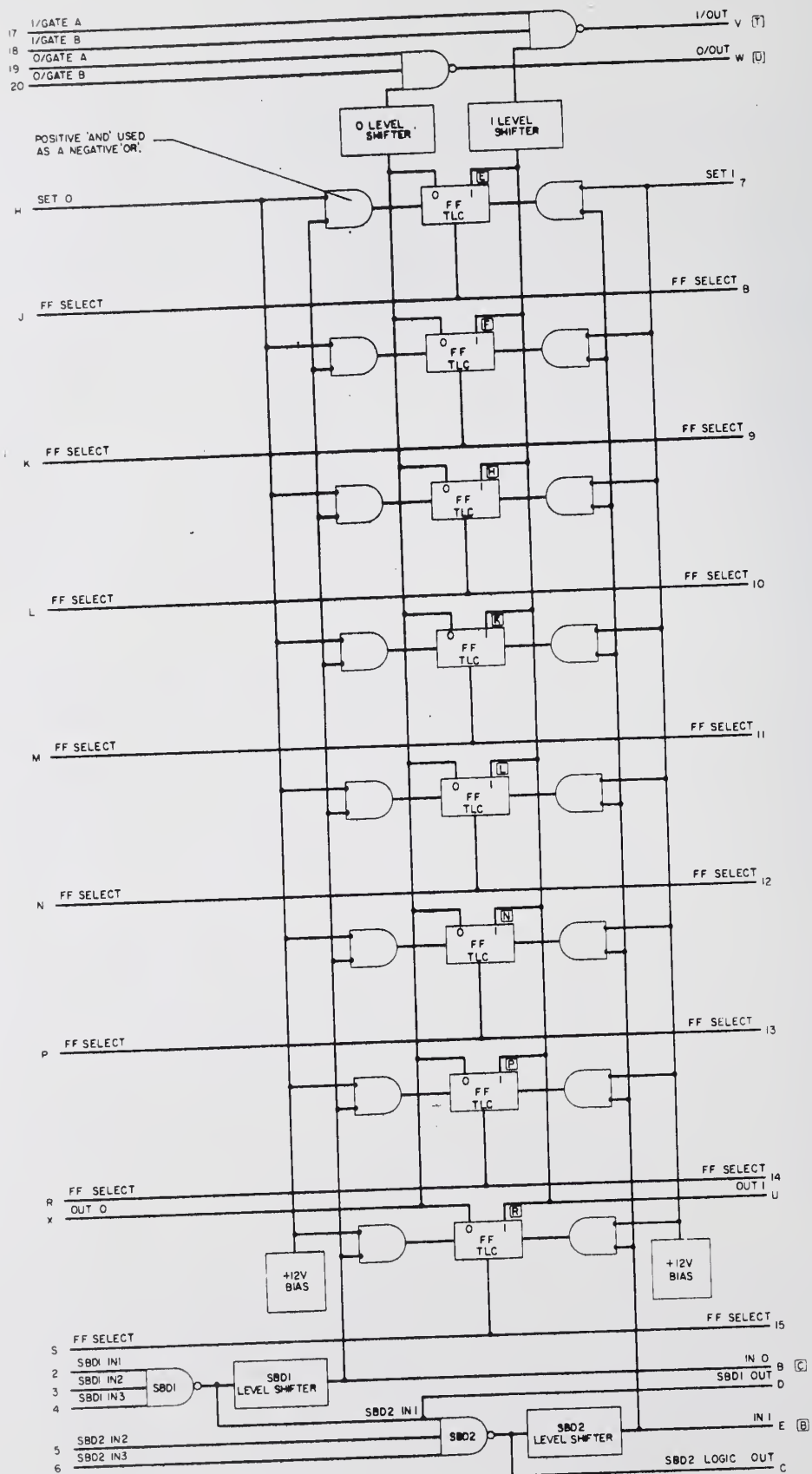


Figure 2.2.4/1 Logic for 204 Board

10/8/69

Section 2.2.4/1 - 2/9

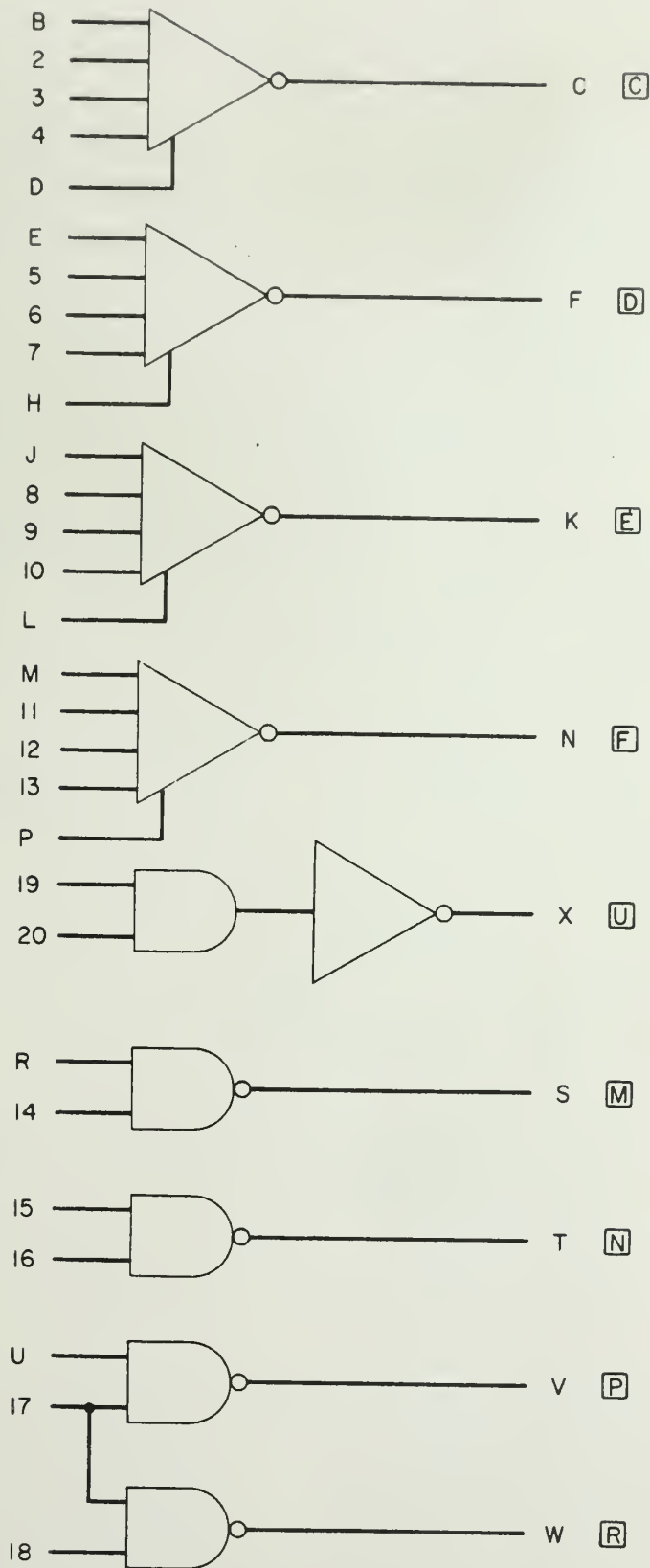


Figure 2.2.4/2 Logic for 205 Board

## NOTES

1. UNLESS OTHERWISE SPECIFIED, RESISTANCE VALUES ARE IN OHMS, 1/4 WATT, 2%.
2. UNLESS OTHERWISE SPECIFIED, DIODES ARE US03.
3. UNLESS OTHERWISE SPECIFIED, TRANSISTORS ARE USN3.
4. UNLESS OTHERWISE SPECIFIED, CAPACITANCE VALUES ARE IN MICROFARADS.
5. TEST POINTS ARE INDICATED BY  $\odot$ , WITH TEST PINS INDICATED BY  $\square$ .
6. UNUSED PINS:  
TEST PLUG—B, H, J, K, L, S, T.
7. POWER SUPPLY COMMON TO 4 CIRCUITS (SIGNAL GOES TO NEGATIVE VOLTAGE LEVEL).
8. COMMON TO 2 CIRCUITS.
9. OUTPUT FROM C, F, K, N IS SPECIAL 3 LEVEL SIGNAL USED TO DRIVE 204 CARD.
10. OUTPUT FROM X, S, T, V, W ARE USED TO DRIVE 204 CARD.

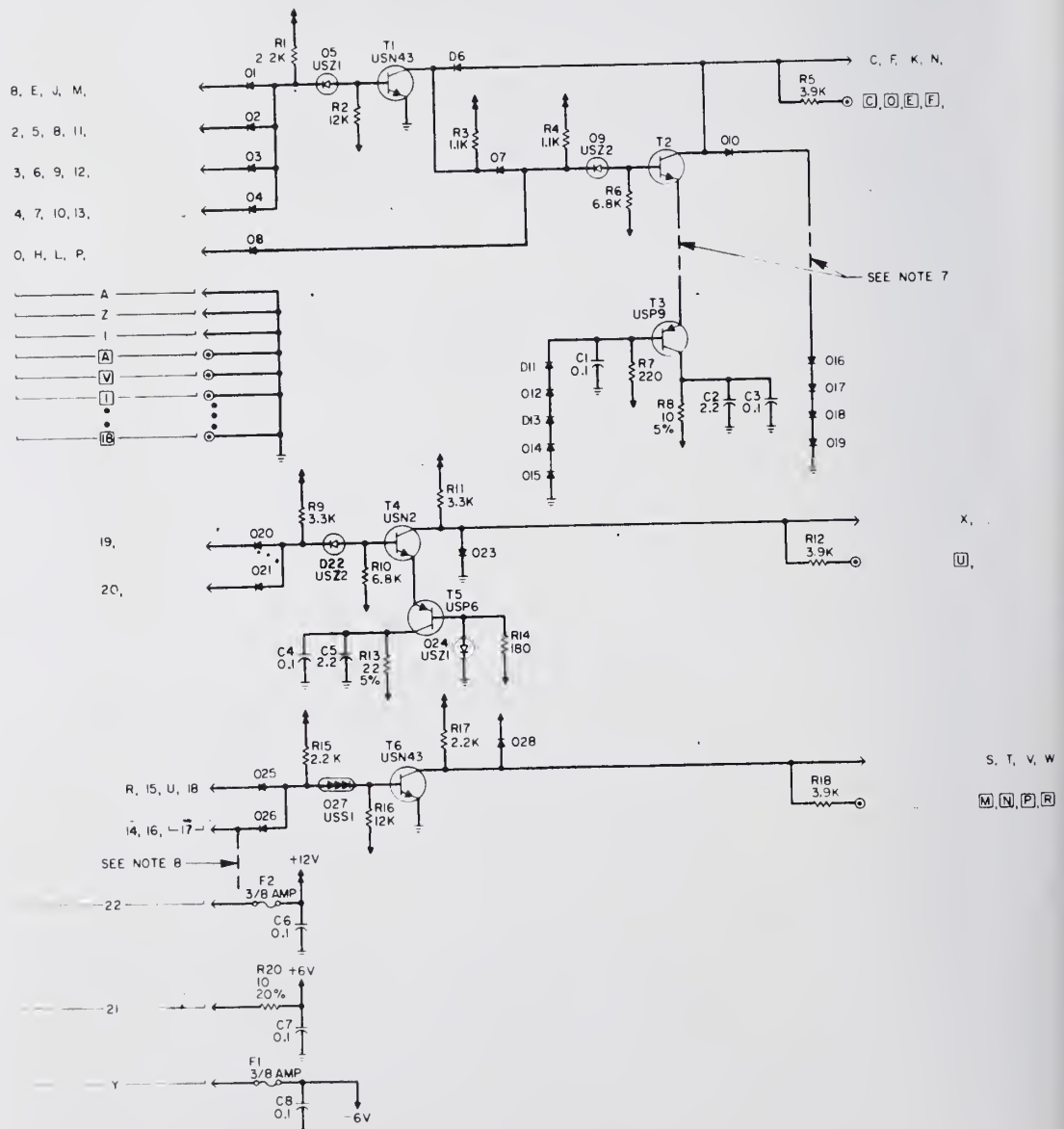


Figure 2.2.4/3 Electrical Schematic for 205 Board



Once the 205 boards have generated the proper voltage levels the 204 board can take them and convert them to read and write operations. Referring to Figure 2.2.4/4 notice that the second circuit from the top has connections to pins J through S, and 8 through 15. These eight circuits are the flip-flops that store the information, and the lines connecting to the above pins are known as the "FF Select" lines (see Figure 2.2.4/1). These FF Select lines have the three levels mentioned above.

Each of these lines connects to the common emitter connection of the flip-flop. These flip-flops were designed so that they will keep their state even if the common emitter voltage varies over a range of  $\pm 4$  volts from ground. Due to some external connections to the flip-flop, however, the mode of operation of the flip-flop changes as the voltage is varied over this range. Specifically, the flip-flops are at rest when the voltage is +1 volt, they are ready to accept new information when the voltage is +3.5, and their internal states are sensed at -3.0 volts.

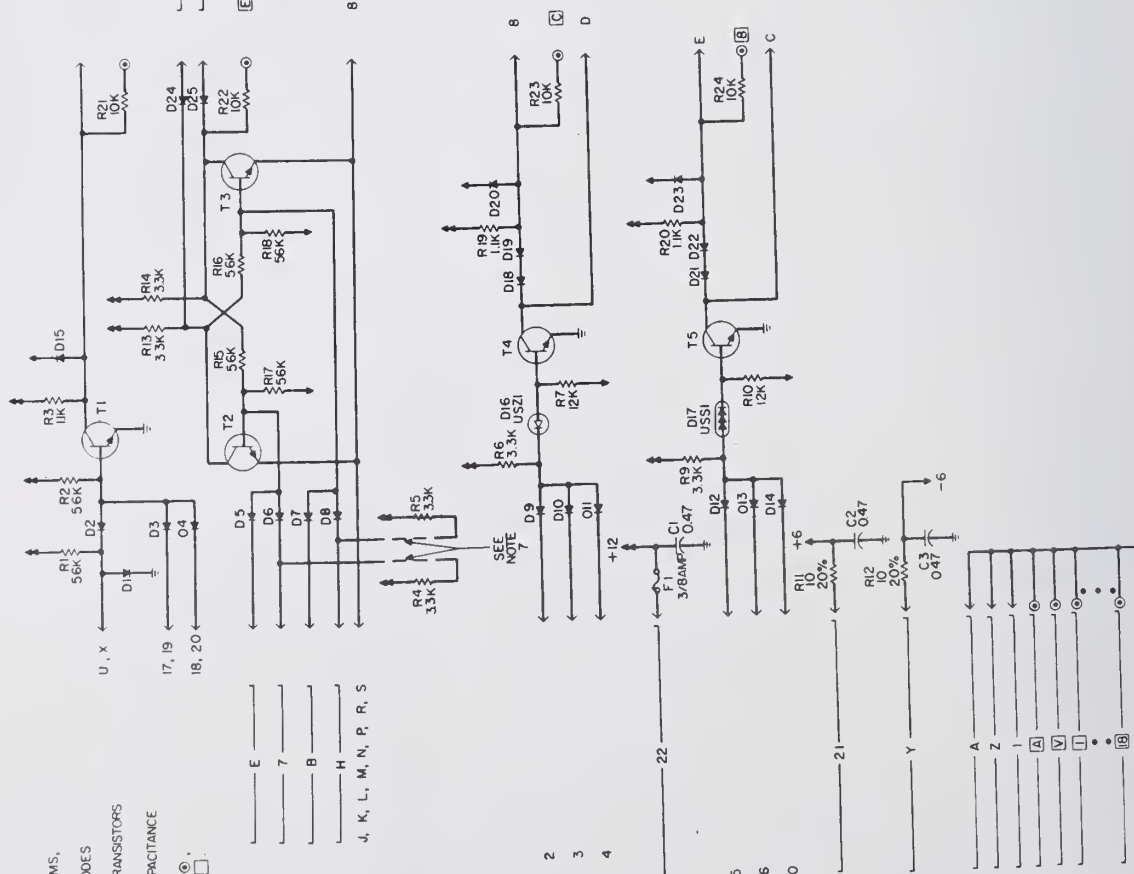
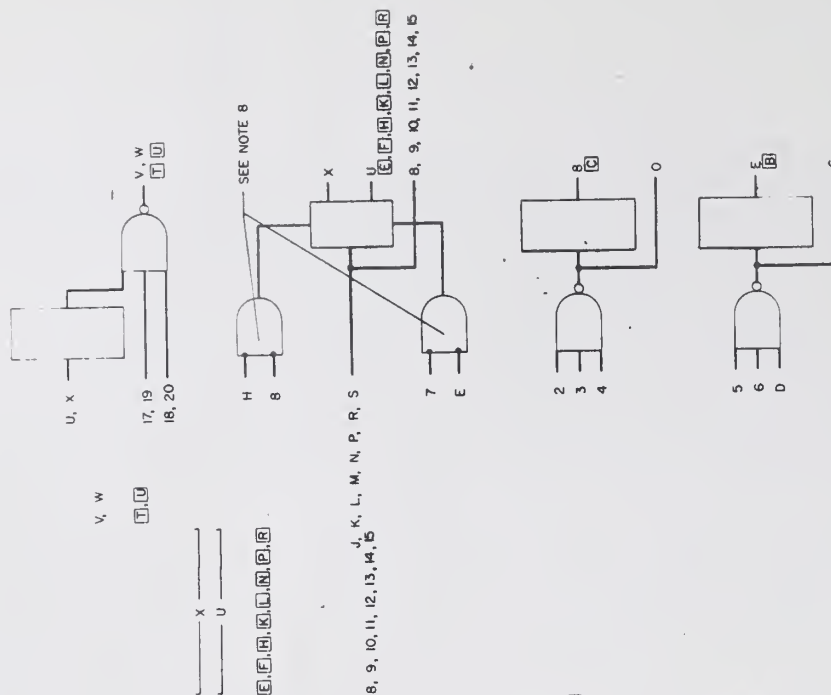
The reason why they act in this manner is not too difficult to understand. Take first the case of new information coming into the flip-flop. When the write gating signal comes along the FF Select lines are placed at +3.5 volts, and the flip-flops will be forced into a raw state if any of the base connected diodes (D5 to D8) are grounded or near ground. If D5 or D6 is grounded, the flip-flop will set to "0", and if D7 or D8 is grounded, the flip-flop stores a "1." A "1" state is defined when the signal at the output test point resistor (R22) is positive.

Diodes D5 and D7 on all flip-flops are connected internally to the IN 1 and IN 0 lines respectively. These two lines are normally the complements of each other, and can never be grounded at the same time. The status of these two lines depends on the inputs to the Storage Bus Drivers (SBD1 & SBD2), and since both SBD gates can be controlled by the inputs to SBD1, complete control over setting a 0 or 1 is possible



- 1 UNLESS OTHERWISE SPECIFIED,  
RESISTANCE VALUES ARE IN OHMS,  
1/4 WATT, 2%
- 2 UNLESS OTHERWISE SPECIFIED, DIODES  
ARE US01
- 3 UNLESS OTHERWISE SPECIFIED, TRANSISTORS  
ARE USN2
- 4 UNLESS OTHERWISE SPECIFIED, CAPACITANCE  
VALUES ARE IN MICROFARADS
- 5 TEST POINTS ARE INDICATED BY 
- 6 UNLESS TEST PINS INDICATED BY 

RACK PLUG—F, T, 16  
TEST PLUG—D, J, M, S  
7 COMMON TO 8 CIRCUITS  
8 POSITIVE 'AND' USE AS A  
NEGATIVE 'OR'.



Section 2.2.4 - 6/9



by using only one input line (SBD1 IN 1, say). A closer look at the output of one of the SBD units shows that these lines actually rest slightly above ground, due to the action of the level shifter D18 & D19 or D21 & D22. These diodes are necessary to decouple the IN 1, IN 0 lines from the SBD outputs while the flip-flops are in their rest states.

The Set 1 and Set 0 lines are similar in function to the IN 1 and IN 0 except that they must be controlled by external logical circuitry. As before, a "1" will be stored when there is a positive signal on Set 1 and a grounded signal on Set 0. When not in use the 12 volt bias resistor assures that these lines are up and out of harm's way at all times. If these lines are controlled by external circuitry, then this circuitry must insure that these lines are positive except during a write command. Otherwise loss of information might occur. These two lines can also be used to override the normal write command and clear the memory of all information. This is accomplished by pulling the Set 1 line down to about -3 volts and leaving the Set 0 positive.

As was mentioned before the SBD gates control the setting of 1's and 0's. It might appear, after a quick glance at the schematic drawing, that these two units are independent of each other and can be wired at will. This is not the case, for both are internally wired to pin D, and consequently to each other. This connection is evident in the logic drawing. This series connection is one reason for the use of a codistor in SBD2. This codistor speeds up the response of T5 to a change in state of T4.

Although not absolutely necessary, it is good practice to supply the IN 0 and IN1 lines with a gating signal. This can be easily accomplished by wiring a pair of the SBD inputs together, say SBD1 IN 3 and SBD2 IN 3. When this combined line is grounded, the outputs from both SBD's will be positive, and IN 0 and IN1 will be up and out of the way. Drivers for this operation are provided on the 205 board.

Now let us consider the case in which information leaves the flip-flops. The easiest way to understand the read operation is to start at the output inverter and work towards the flip-flop. Notice first that if any of the signals appearing on terminals 17, 18, 19, of 20 is below ground, then T1 is unconditionally off, and the output signal will be positive. When T1 is unconditionally off signals appearing at X and U have no effect whatever on the output of the transistor. Thus, lines 17-20 serve as gating lines controlling the output buffer. The 205 card supplies these negative going gating signals.

If none of the above lines are at or below ground, then the state of T1 depends on the signals present at U or X. The normal resting voltage for these lines is about +.8 volts, and this also is the maximum voltage possible due to the forward bias across D1. When these lines are pulled down below ground T1 again turns off, and the output is positive. Finally we can see why pulling down on the FF Select lines causes these flip-flops to be read. One of the two transistors in the flip-flop will be saturated, causing one of the output diodes D24 or D25 to pull down lines X or U. This completes the chain of events.

Simultaneous Read-Write is possible because the flip-flop in the read mode will not interfere with the flip-flop in the write mode.

The initials TLC on the flip-flop (Figure 2.2.4/1) indicate that this input is for Three-Level-Conversion as opposed to a trigger input for the flip-flop.

With proper timing relations it should be possible to complete a read or write within 100 nsec.

# Summary: Rack Plug Signal Descriptions

<u>Pins</u>	<u>Name</u>	<u>Normal Use</u>
B	IN 0	Check status of IN 0 bus.
C	SBD2 Out	Checks collector of T5
D	SBD1 Out	Checks status of SBD1 transistor, input to SBD2
E	IN 1	Check status of IN 1 bus
H	SET 0	Input connection for setting 1 or setting all 1's*
J-S, 8-15 FF Select Three level input lines to FF. Note: Normal wiring of these boards will have 8 of A going to J of B, 8 of B going to J of C, and so on for adjacent boards A, B, C.		
U	Out 1	Checks status of Out bus.
V	1/Out	Output connection from buffer
W	0/Out	Output connection from buffer
X	Out 0	Checks status of Out bus
2,3,4	SBD1 IN's 1,2,3	Input connection for 1/0 set, gating
5,6	SBD2 IN's 2,3	Input connection for gating only
7	SET 1	Input connection for clearing or setting 0*
17,18	1/Gate	Input gate connections for output buffer
19,20	0/Gate	Input gate connections for output buffer

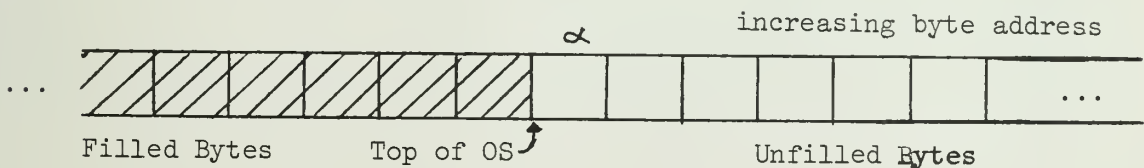
\*This apparent confusion in terminology arises from the fact that these gates are positive logic AND's used as negative logic OR's.



### 2.3 Operand Stack

The Operand Stack (OS) is a specially designated file in the main store defined by pointer register #13. It gains its distinction by virtue of having its topmost bytes (up to a maximum of 32) located in fast access storage registers internal to the Taxicrinc Processor. For this reason the Operand Stack can be used as a scratch pad for evaluating arithmetic and boolean expressions with a minimum access to core storage. Many TP instructions obtain operands from or operate directly on the OS. Some instructions, such as LD or PØP, merely transfer data cells into or out of the OS. Others, like ADD, pop out two cells, operate upon these operands and then return a single result.

Programming conventions established for the OS make it appear that the stack is at all times entirely in core. It can be visualized as a continuous string of bytes. The Operand Stack Pointer (Pointer Register #13) can be considered to contain the 16 bit file address  $\alpha$  of the first (leftmost) unfilled byte in the stack, relative to its associated base.



An operand can be pushed into the stack by assigning it to the field starting at location  $\alpha$ , and then incrementing the OS pointer by the field (or cell) size. In like manner, an operand may be popped from the stack by decrementing the OS pointer by the field length (i.e. 1, 2, 4 or 8 bytes). Operand boundaries are by no means sacred: for

example, fields pushed into the stack as bytes may be popped out a word at a time.

To add or delete from the top of the stack we must:

- 1) specify the field length of the operand by giving a "cell size" parameter as part of the instruction. The cell size parameter consists of two bits and identifies the operand as a byte (00), halfword (01), word (10) or double word (11).
- 2) distinguish whether the operand is to be pushed into the stack (at the address = the top of the stack) or popped from the stack (at the address = the top of the stack minus the cell size).

Although the top bytes of the Operand Stack are located in fast registers in the TP, any operand in the stack may be accessed by using PR #13 as a file address and suitably modifying it to get the desired byte within the stack. As will be seen in Section 4.2.1.2, this request necessitates emptying the contents of the TP fast registers into core first. The complete stack will then indeed be in core, as the programmer expects. Of course this process involves time; however it is not anticipated that this method of accessing the Operand Stack will be used often, so this slowdown should not be significant.

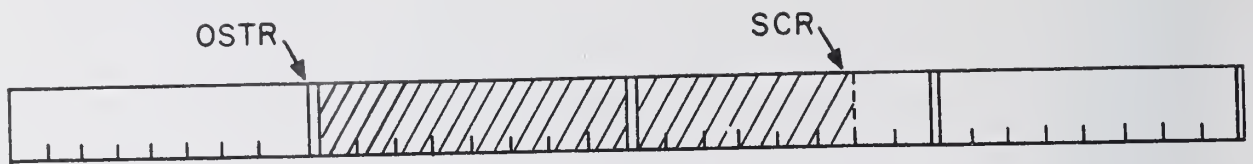
Each byte of the Operand Stack has been assigned a unique "wired in" address, 0 through 31 (00000 through 11111). A five bit register, the Stack Control Register (SCR), holds the address of the top of the stack: the top is defined as the first empty byte. This register is changed whenever a new operand is added to the OS. Since it would take excessive time to always access PR #13 to find



out the top of the stack, PR #13 is not kept current! What is done is the following: Whenever PR #13 is accessed, the SCR is masked into its 5 lower order bit positions, thus giving the true top position. Thus the only time the OS control must change PR #13 is when there is overflow or underflow in the highest order bit of the SCR. This situation is called "stack wraparound" since it occurs when the SCR "moves" across the 31-0 boundary.

As described above, the topmost bytes of the OS are located in up to 32 bytes of fast storage in the TP. These 32 bytes of fast storage are equivalent to four double words. A unique address for each byte position can be determined from five bits. The high-order two bits define which of the four double words contains the top of the OS. The four double words can hence be assigned the "addresses" 00000, 01000, 10000, and 11000.

The Operand **Stack Top Register (OSTR)**, a 2-bit register, contains the double word address (00, 01, 10, 11) of the boundary double word, i.e. the double word which has been in the stack for the longest time. Expressed in a different manner, the OSTR marks the boundary between the part of the OS contained in the TP hardware and the part of the OS contained in main memory. The use of the SCR and OSTR registers is shown in Figure 2.3.



0	1

 OSTR

1	0	1	1	0

 SCR

In this example the shaded bytes contain useful data. The second double word (01) has been in the stack the longest time. Additions or deletions to the stack are made beginning at the location specified by the SCR.

Figure 2.3 Organization of the Operand Stack



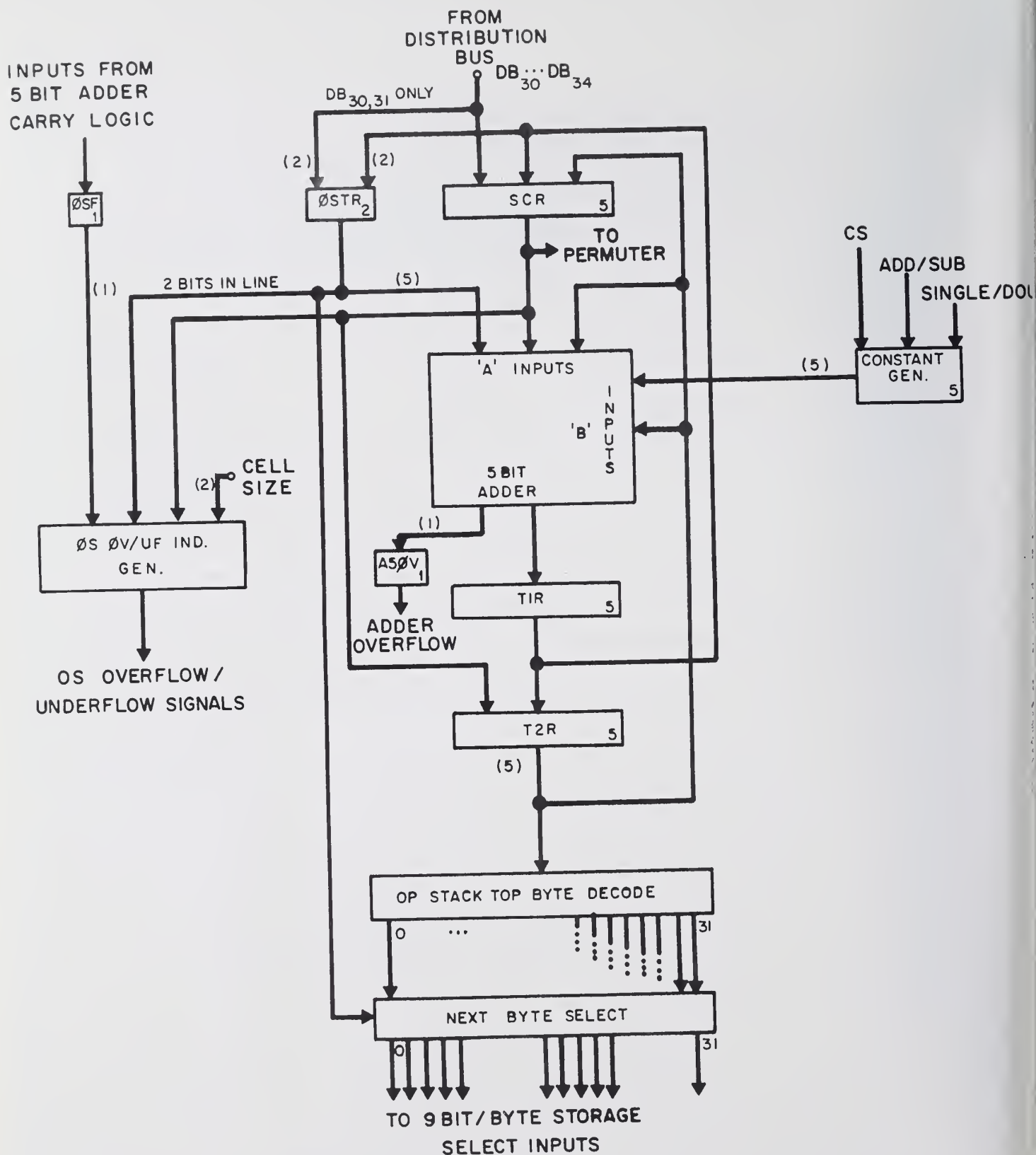
### 2.3.1 Operand Stack - Functional Description

#### 2.3.1.1 Operand Stack Control Registers

The hardware organization of the OS components is shown in Figure 2.3.1.1. There are four registers associated with the operand stack: the Operand Stack Top Register OSTR and the Stack Control Register SCR mentioned above and the temporary registers T1R and T2R. All registers except the OSTR are five bits long.

The OSTR (Operand Stack Top Register) holds the double-word address - one of four - of the "bottom" of the OS in the TP's hardware storage. Referring to Figure 2.3 this boundary may alternately be considered the upper bound of the stack in the hardware registers, since the TP treats the hardware stack storage in a "circular" manner, e.g., if the 31st byte is filled and more storage is needed, the 0th byte is used. Beyond this bound, information will be overwritten and destroyed unless cells are first taken from the "bottom" of the hardware stack and stored in main memory.

The SCR (Stack Control Register) contains the address (one of 32) of the first unfilled byte beyond the top of the OS. The contents of the SCR are incremented and decremented by the appropriate cell size whenever a cell is accessed or loaded into the OS. The SCR itself is loaded anytime PR#13 is loaded; and when the contents of PR#13 are gated out, SCR's contents are masked into the five low order bits of the PR in order to give the complete address of the top of the OS. If and when SCR overflows (from PUSH's) or underflows (due to POP's), PR#13 is incremented(or decremented)by 32 to absorb the over/underflow.



OPERAND STACK CONTROL LOGIC SYSTEM  
FIGURE 2.3.1.1

### 2.3.1.2 The Constant Generator

One of the 'B' inputs to the adder, as stated above, is the 5 bit constant generator output bus, Ki. The constant generator has as inputs 1) the cell size (CS) signals, 2) the add/subtract signal, 3) a single/double or multiply signal, and 4) several control lines used for generating specific constants. It has as an output, a 2's complement number, which appears on the output bus, and which equals  $\pm$  one or two times the number of bytes in the input cell size. The outputs are  $\pm 1$ ,  $\pm 2$ ,  $\pm 4$ ,  $\pm 8$  and  $\pm 16$ , depending on the inputs. If, for example, the add/subtract input = 1 (add), the multiplicity signal = 1 (multiply by 1), and CSH = 1, the output is +2. If the multiply line = 0 (multiply by 2), the output is +4.

The constant generator operates using an intermediate 5 bit bus,  $\overline{\text{CGBi}}$ , on which the magnitude of the desired number is generated. The number is in its complemented form at this point. The  $\overline{\text{CGBi}}$  bus is then gated to the output bus either with or without first being complemented depending on whether the positive or negative form of the constant is desired.

Referring to the truth table in Figure 2.3.1.2, the following equations can be written for the intermediate bus:

$$\overline{\text{CGB1}} = \overline{\text{X2}} \cdot \overline{\text{CSD}} \cdot \overline{\text{OSC16/E}}$$

$$\overline{\text{CGB2}} = \overline{\text{X1}} \cdot \overline{\text{CSD}} \cdot \overline{\text{X2}} \cdot \overline{\text{CSW}} \cdot \overline{\text{OSC8/E}}$$

$$\overline{\text{CGB3}} = \overline{\text{X1}} \cdot \overline{\text{CSW}} \cdot \overline{\text{X2}} \cdot \overline{\text{CSH}} \cdot \overline{\text{OSC4/E}}$$

$$\overline{\text{CGB4}} = \overline{\text{X1}} \cdot \overline{\text{CSH}} \cdot \overline{\text{X2}} \cdot \overline{\text{CSB}} \cdot \overline{\text{OSC2/E}}$$

$$\overline{\text{CGB5}} = \overline{\text{X1}} \cdot \overline{\text{CSB}} \cdot \overline{\text{OSC1/E}}$$

where the OSCi/E signals are the control lines used for generating specific constants which were mentioned earlier and X1 and X2 are the multiply by 1 and multiply by 2 signals, respectively.

The final stage in the generator consists of gating this signal to the output bus,  $\overline{Ki}$ . If the number is to be negative, the  $\overline{CGBi}$  signals are gated directly. If the number is to be positive, the  $\overline{CGBi}$  signals are first inverted and then gated to the  $\overline{Ki}$  bus.

The actual control signals act as follows:

$\overline{X2} = 0$  multiply by 2 (i.e.  $X2 = 1$ )

$\overline{X2} = 1$  multiply by 1 (i.e.  $X1 = 1$ )

$\overline{S/E} = 0$  negative number

$\overline{S/E} = 1$  positive number

In order to generate constants using the special control signals, both X1 and X2 must be set to zero (see previous equations). This is accomplished by setting  $\overline{X2}$  to "0" and setting a special control signal,  $\emptyset SCG/E$ , to "1". Since the inverse of  $\emptyset SCG/E$  is dot-or'ed to the inverse of X2, both the X1 and X2 gates will be "0", and only the control signals OSCi/E will be able to determine the state of  $\overline{CGBi}$ .

	CGB <sub>1</sub>	CGB <sub>2</sub>	CGB <sub>3</sub>	CGB <sub>4</sub>	CGB <sub>5</sub>	CGB <sub>1</sub>	CGB <sub>2</sub>	CGB <sub>3</sub>	CGB <sub>4</sub>	CGB <sub>5</sub>
CSB	0	0	0	0	1	0	0	0	1	0
CSH	0	0	0	1	0	0	0	1	0	0
CSW	0	0	1	0	0	0	1	0	0	0
CSD	0	1	0	0	0	1	0	0	0	0

X1(multiply by 1)

X2(multiply by 2)

	CGB <sub>1</sub>	CGB <sub>2</sub>	CGB <sub>3</sub>	CGB <sub>4</sub>	CGB <sub>5</sub>
OSC16/E	1	0	0	0	0
OSC8/E	0	1	0	0	0
OSC4/E	0	0	1	0	0
OSC2/E	0	0	0	1	0
OSC1/E	0	0	0	0	1

Note: Only one of CSB, CSH, CSW and CSD can be on at one time.

Figure 2.3.1.2



### 2.3.1.3 Byte Selection Logic

In order to know where a cell is to be stored or read from in the 32 byte buffer storage of the operand stack, a method is needed to select the bytes desired for any current action. This is the purpose of the byte selection logic. The basic idea involved is that the bytes concerned should first be chosen on the basis of the single byte address in register T2R and then this byte and the necessary bytes after it can be either read out of or written into the  $\emptyset S$ .

The four different cell sizes cause a problem of picking out the right number of bytes. However all data lines are built for four bytes therefore it was decided that four bytes would be selected regardless of cell size. For word or double-word cell size (a double word size needs two "accesses" to the stack anyway because of the size of the data lines) this procedure is optimal. The half-word and byte cell sizes also usually work because only the top of the stack is ever accessed and if the cell desired is only 1 or 2 bytes long, the other 3 or 2 bytes will contain garbage which can be ignored. The only exception to this rule occurs when the  $\emptyset STR$  boundary occurs within 3 or 2 bytes of the last byte which needs to be accessed. Referring to Figure 2.3.1.3, for example, if the rightmost half word in the 00 double word were the cell being written on, and the selection logic picked out four bytes, the first two bytes in the 01 double word would be overwritten and destroyed. For this reason the byte selection logic does not allow four bytes to be chosen in such a manner that the  $\emptyset STR$  boundary will be "crossed".

With this in mind the byte decoding and selection logic is relatively straightforward. The byte address in T2R is decoded to give one of 32 outputs. Each output represents a particular byte in the hardware  $\emptyset S$ . In the next-byte-selection logic the next three bytes in increasing order are selected along with the original byte and selection signals are generated



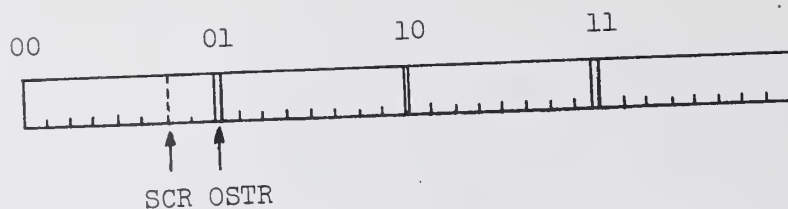


Figure 2.3.1.3

to be delivered to the actual hardware  $\emptyset S$ . These selection signals may wrap around the "11111" - "00000" boundary or cross any other double word boundary except the boundary defined by the  $\emptyset STR$ . In this case the selection of bytes past the boundary is inhibited.

The boundary inhibit gates are inserted every eight bytes. The one (of four) boundary gates which inhibits propagation of select-next signals is itself selected by the (decoded) contents of the  $\emptyset STR$ . Boundaries are located at the four "natural" wired-in double-word addresses. The inhibit gates allow cells to be loaded into the  $\emptyset S$  right up to, but never beyond, the boundary.



#### 2.3.1.4 Stack Overflow-Underflow Sensing Logic

The stack overflow/underflow sensing logic determines the position of the top of the stack within the 32 bytes in OS hardware buffers relative to the boundary between the TP and Main Storage portion of the stack. Inputs to the OV/UF logic come from the OSTR, the SCR, the adder overflow flip-flop A50V and the cell size decoder.

The philosophy of the OV/UF logic is first to discover which double word the top of the stack (SCR) is in relative to the boundary (OSTR): the first or second double-word above the boundary or the first or second below the boundary. Then, on the basis of the input cell size and the position of the top of the stack within the double-word (as given by the low-order bits of the SCR) the appropriate output signal is activated: OV1, OV2, UF1, or UF2. OV1 indicates that there is inadequate room in the hardware stack to insert one cell of the indicated size. OV2 indicates that there is not room for two of the given cells. UF1 states that there remains less than one of the cell of the indicated size in the hardware stack. And UF2 shows there is less than two cells remaining.

The equations for the position of the top of the OS relative to the boundary are:

$$E\emptyset 1 = \overline{X}_1 \overline{X}_2 \overline{Y}_1 Y_2 \vee \overline{X}_1 X_2 Y_1 \overline{Y}_2 \vee X_1 \overline{X}_2 Y_1 Y_2 \vee X_1 X_2 \overline{Y}_1 \overline{Y}_2$$

= '1' if the top of the OS is in the D-word immediately preceding the boundary.

$$E\emptyset 2 = X_1 \overline{X}_2 \overline{Y}_1 \overline{Y}_2 \vee X_1 X_2 \overline{Y}_1 Y_2 \vee \overline{X}_1 \overline{X}_2 Y_1 \overline{Y}_2 \vee \overline{X}_1 X_2 Y_1 Y_2$$

= '1' if the top of the OS is in the second D-word preceding the boundary.

$$EU1 = \overline{X}_1 \overline{X}_2 \overline{Y}_1 \overline{Y}_2 \vee \overline{X}_1 X_2 \overline{Y}_1 Y_2 \vee X_1 \overline{X}_2 Y_1 \overline{Y}_2 \vee X_1 X_2 Y_1 Y_2$$

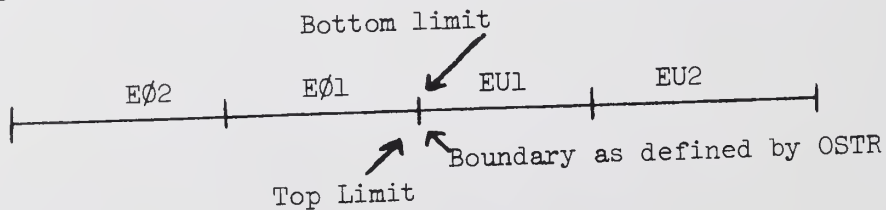
= '1' if the top of the OS is in the same D-word as the boundary.

$$EU2 = \overline{X}_1 \overline{X}_2 Y_1 Y_2 \vee \overline{X}_1 X_2 \overline{Y}_1 \overline{Y}_2 \vee X_1 \overline{X}_2 \overline{Y}_1 Y_2 \vee X_1 X_2 Y_1 \overline{Y}_2$$

= '1' if the top is in the second D-word above the boundary.

In the above equations,  $X_1$  and  $X_2$  are defined as  $SCR_1$  and  $SCR_2$  and  $Y_1$  and  $Y_2$  are  $OSTR_1$  and  $OSTR_2$  respectively.

The above overflow/underflow enable signals are then combined with the cell size and SRC inputs to generate the true OV/UF signals. The "region of effect" of the E-signals may be shown by the following diagram:



In the case where the SCR points to the boundary defined by the  $\emptyset$ STR we have a problem of knowing whether the stack is completely empty or completely full. For this situation  $EU1 = 1$ , while  $EU2$ ,  $E\emptyset1$  and  $E\emptyset2 = 0$ . What we really want to know, is where the SCR was pointing just before it moved to the  $\emptyset$ STR boundary. If it was in the  $EU1$  region, the stack is now empty, while if it was previously in the  $E\emptyset1$  region it is now full.

To determine this, a stack full flip-flop ( $\emptyset$ SF) is used. Whenever the SCR is modified using the SCR M $\emptyset$ D control sequence, a signal sets  $\emptyset$ SF to 1 if the following conditions are satisfied:

- a) the first two bits of the new value of the SCR (which at this time will be in the adder output register, TR1) are equal to the  $\emptyset$ STR,
- b) the low order 3 bits of the SCR are zero,
- c) the  $E\emptyset1$  signal was activated by the old value of the SCR.

This assures that when the SCR "approaches" the  $\emptyset$ STR from region  $E\emptyset1$ , the stack will be declared to be full. Once  $\emptyset$ SF has been turned on, it is used to keep both  $\emptyset$ U1 and  $\emptyset$ U2 turned on.

$\emptyset$ SF is reset by the main control using the  $\overline{R\emptyset SF}$  signal whenever the SCR is modified. This is necessary as a preliminary to checking for a possible "stack full" condition at the new SCR position.  $\emptyset$ SF is also reset whenever the  $\emptyset$ STR is moved.

The actual overflow/underflow signals may now be defined:

$$OV1 = \emptyset SF \vee E\emptyset1 \cdot CSD \cdot (X_3 \vee X_4 \vee X_5) \vee E\emptyset1 \cdot CSW \cdot X_3 \cdot (X_4 \vee X_5) \\ \vee E\emptyset1 \cdot CSH \cdot X_3 \cdot X_4 \cdot X_5$$

$$OV2 = \emptyset V1 \vee E\emptyset2 \cdot CSD \cdot (X_3 \vee X_4 \vee X_5) \vee E\emptyset1 \cdot CSW \cdot (X_3 \vee X_4 \vee X_5) \\ \vee E\emptyset1 \cdot CSH \cdot X_3 \cdot (X_4 \vee X_5) \vee E\emptyset1 \cdot CSB \cdot X_3 \cdot X_4 \cdot X_5$$

$$UF1 = EU1 \cdot CSD \vee EU1 \cdot CSW \cdot \overline{X}_3 \vee EU1 \cdot CSH \cdot \overline{X}_3 \cdot \overline{X}_4 \vee EU1 \cdot CSB \cdot \overline{X}_3 \cdot \overline{X}_4 \cdot \overline{X}_5$$

$$UF2 = UF1 \vee EU2 \cdot CSD \vee EU1 \cdot CSW \vee EU1 \cdot CSH \cdot \overline{X}_3 \vee EU1 \cdot CSB \cdot \overline{X}_3 \cdot \overline{X}_4$$

where once again the X's refer to the SCR bit positions.

These signals are used in the OS sequencing control to control the stack accessing and updating and to control the time of transfer of overflow/underflow double words between the hardware and core portions of the OS.

Temporary Register #1 (T1R), as the name implies, is used to hold temporarily a five-bit stack address. T1R has the output from the five-bit adder as its only input. Its primary function is to receive a sum from the adder before passing it on to one of the other registers. Bits 1 and 2 are also employed in the overflow/underflow logic, discussed above.

Temporary Register #2 (T2R) is, again, used to temporarily hold five-bit stack addresses. However, T2R's functions are less transient than those of T1R. The purpose of T2R is, given a five-bit stack address, to drive the decoding and byte-selection logic, which in turn drives the  $\emptyset$ S word select inputs. T2R has as inputs the outputs of T1R and the SCR. T2R's outputs are also fed to the SCR.

Two temporary registers are not essential for the proper operation of the  $\emptyset$ S control logic. They do offer a speed increase over one-register operation. For example, if two cells are to be accessed in quick succession, the address of the first may be constructed in the five-bit adder and gated into T2R via T1R or gated directly into T2R from the SCR. Then, while the first cell is being accessed, the address of the second is computed and placed into T1R ready to be gated into T2R as soon as the access of the first cell is complete.

The adder itself uses full carry lookahead to speed addition. It adds in fewer than 6 NAND delays. (The reader is referred to the writeup of the 32 bit adder in Section 2.4 for a discussion of the carry lookahead principle and equations.) Its inputs are classified as 'A' and 'B' inputs. The sum is obtained from one 'A' plus one 'B' input. 'A' inputs are the SCR,  $\emptyset$ SR, and T2R. 'B' inputs come from the constant generator and T2R. Any 'A' input may be added to any 'B' input. The output of the adder goes to T1R.

If an overflow (or underflow) condition is detected, the five-bit adder overflow flip-flop (A5OV) is set. When this occurs, Pointer Register #13 is incremented (decremented) at an appropriate time determined by the main control.

#### 2.3.1.5 5-Bit Adder-Functional Description

The 5-bit Adder is the small full carry lookahead adder used by the Operand Stack logic to perform any additions which may be needed in its operation. This adder is totally independent and completely separate from the Main TP 32-bit Adder.

The 5 bit Adder has two 5 bit input busses. The A input bus has 3 possible sources: temporary register T2R, the Stack Control Register, SCR, or the Operand Stack Top Register, OSTR. Note that in the latter case only the two highest order bits are used. The B input bus has two possible sources: the Constant Generator (see Section 2.3.1.2) or the **temporary** register T2R.

The output of the 5-bit Adder is sent to temporary register T1R. If an illegal result is obtained due to an overflow on either an addition or a subtraction, the A5OV flip-flop is set.

The 5 bit Adder is generally used to add or subtract numbers from the SCR or OSTR when these registers are "moving" one of the Operand Stack boundaries. The Constant Generator can generate 1's complement numbers and if a low order carry is injected into the adder (C5IN J) at the same time, a subtraction will result.



## 2.3.2 Signal Name Lists for the Operand Stack

### 2.3.2.1 Control Signals

Operand Stack:

CKØF/E	- Enable ØS overflow/underflow check
DBØS/G	- Gate the DBP → ØS according to ØS select lines
DBST/G/	- Gate DB → SCR and DB → ØSTR (part of stack initialization)
FLØS/N	- Inhibit flags of DBP from being gated into ØS storage
ØSPF/G/	- Gate ØS → permuter (using ØS select) 1's complement
ØSPT/G/	- Gate ØS → permuter (using ØS select) true
OSTP/G	- Mask OSTR into permuter via ØSR
ØV1, ØV1/	- Output - indicates no room in ØS for cell of current size
ØV2	- Output - indicates no room in ØS for 2 cells of current size
RØSF/	- Reset "ØS full" flip-flop
SCR P/G	- Mask SCR into permuter via ØSR
SCT2/G/	- Gate SCR → T2R
T1R2/G/	- Gate T1R → T2R
T1SC/G/	- Gate T1R → SCR
T1ST/G/	- Gate T1R → ØSTR
T2SC/G/	- Gate T2R → SCR
UF1	- Output - indicates ØS does not contain 1 cell the size of CS
UF2	- Output - indicates ØS does not contain 2 cells the size of CS.
ZSCR	- Output - SCR(3), SCR(4) and SCR(5) are 0



## 5 Bit Adder:

ADD5/E/	-	Enable the 5 bit adder
KB/G	-	gate output of const. gen. to B input of 5 bit adder
ØTA/G	-	gate ØSTR → A input of the 5 bit adder
RSYS/	-	reset system
SCA/G	-	gate SCR → A input of the 5 bit adder
T2A/G	-	gate T2R → A input of the 5 bit adder
T2B/G	-	gate T2R → B input of the 5 bit adder

## Constant Generator:

X2/	-	if "0", multiply output of constant generator by 2
S/E	-	subtract enable - if = 1, generate negative const.
CGBi/	-	constant generator intermediate bus - bit i
ØSCG/E/	-	enable generation of a const. from control
ØSCG1/E	-	generate 1 on constant bus
ØSCG2/E	-	generate 2 on constant bus
ØSCG4/E	-	generate 4 on constant bus
ØSCG8/E	-	generate 8 on constant bus
ØSCG16/E	-	generate 16 on constant bus



### 2.3.2.2 Internal Signals Used by the Operand Stack

#### Operand Stack

BROSi	-	Input bus to permuter from Base Register and OS storage - bit i
BYi	-	First byte select from $\emptyset S$ , byte i on if it is first byte selected decoded from T2R.
E $\emptyset$ 1	-	= 1 if top of $\emptyset S$ is in DW immediately preceding $\emptyset STR$ boundary.
E $\emptyset$ 2	-	= 1 if top of $\emptyset S$ is in second DW immediately preceding $\emptyset STR$ boundary.
EU1	-	= 1 if top of $\emptyset S$ is in same DW as $\emptyset STR$ boundary.
EU2	-	= 1 if top of $\emptyset S$ is in second DW above $\emptyset STR$ boundary.
$\emptyset SF$	-	Operand Stack Full
$\emptyset Si/S/$	-	$\emptyset S$ select; if on indicates ith byte has been selected (1 to 4 signals will be on at one time)
$\emptyset STRi$	-	Operand Stack Top Register (indicates the double word at the boundary between the hardware OS stack and core memory) - bit i
SCRi	-	Stack Control Register (gives current entry point to OS) - bit i
T1Ri	-	Temporary Register #1 = output from 5 bit adder - bit i
T2Ri	-	Temporary Register #2 = output from T1R or SCR - bit i

## 5 Bit Adder:

Ai	-	"A" input to Adder (= ØSTR, SCR, T2R) - bit i
A5ØV	-	if on, 5 bit adder overflowed
Bi	-	"B" input to Adder (= T2R, K) - bit i
Ki	-	output of const. generator - bit i
OSTRi	-	"Operand Stack Top" register - bit i
SCRi	-	Stack Control Register - gives current entry point to OS - bit i
T1Ri	-	Temporary Register #1 - 5 bit adder output - bit i
T2Ri	-	Temporary Register #2 - bit i

## Constant Generator:

CSB	-	Cell size is byte
CSD	-	Cell size is double word
CSH	-	Cell size is halfword
CSW	-	Cell size is word
Ki	-	Output of const. generator - bit i

```

// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  OPSTKIN: PROC(CKOFE,      CSB,      CSD,      CSH,      CSW,
                DBOSG,      OSF,      OSPFG,      OSPTG,      OSSS,
                OV1,      OV2,      ROSF,      UF1,      UF2,
                ZSCR);
  DCL(CKOFE,CSB,      CSD,      CSH,      CSW,      DBOSG,
      OSF,      OSPFG,      OSPTG,      OSSS,      OV1,      OV2,
      ROSF,      UF1,      UF2,      ZSCR) BIT(1);
  DCL( BRDS(36), BY(0:31), OS(0:31,9), OSS(0:31), OSTR(2),
      SCR(5), T1R(0:5),T2R(5)) BIT(1) EXTERNAL;

  DCL
    (X(5),
     Y(2)
    ) BIT(1),
    (I,J,L) FIXED BIN,
    (EO1,
     EO2,
     EU1,
     EU2
    ) BIT(1);
  DCL T2RVAL FIXED BIN;
  DCL OSTRVAL FIXED BIN;

  /* USE BYTE DECODER*/
  /* USE THE CONTENTS OF T2R TO SELECT THE CORRESPONDING
  BY(I) SIGNAL */

  OSS='0'B;
  BY='0'B;
  OSSS=DBOSG|OSPTG|OSPG;
  IF ~OSSS THEN GO TO CKSCRZ;

  T2RVAL=0;
  DO I=1 TO 5;
    IF T2R(I) THEN
      T2RVAL=T2RVAL+2** (5-I);
  END;
  BY(T2RVAL)='1'B;

  /* NEXT 3 BYTE SELECT.
  USE THE BY(I) SIGNALS TO SELECT AN OSS(I) SIGNAL AND UP TO
  3 SUCCESSIVE OSS(I) SIGNALS */

  IF T2RVAL<=28 THEN
    DO I=T2RVAL TO (T2RVAL+3);
      OSS(I)='1'B;
    END;

  ELSE DO;
    DO I=T2RVAL TO 31;
      OSS(I)='1'B;
    END;

```

```

        DO I=0 TO (T2RVAL-28);
            OSS(I)='1'B;
        END;

    END;

    OSTRVAL=0;
    IF OSTR(1) THEN OSTRVAL=OSTRVAL +16;
    IF OSTR(2) THEN OSTRVAL=OSTRVAL +8;
    IF OSTRVAL=0 THEN DO;
        IF BY(29)|BY(30)|BY(31) THEN OSS(0),OSS(1),OSS(2)='0'B;
    END;
    ELSE DO;
        I=OSTRVAL;
        IF BY(I-1)|BY(I-2)|BY(I-3)
            THEN OSS(I),OSS(I+1),OSS(I+2)='0'B;
    END;

    /* LOAD BROS FROM OS */

    IF (OSPTG|OSPFG)&-DBROSG THEN DO;

        DO I=0 TO 31;
            IF OSS(I) THEN DO;
                L=MOD(I,4);
                /* L IS THE BYTE POSITION OF THE BROS TO WHICH THE
                   SELECTED BYTE OF THE OS IS GATED */
                DO J=1 TO 9;
                    BROS(L*9+J)=OS(I,J);
                END;
            END;
        END;

        IF OSPFG THEN BROS =-BROS;
    END;

    /*CHECK FOR SCR ON DOUBLE WORD BOUNDARY, I.E. LOW ORDER 3
       BITS =0 */

    CKSCRZ:  ZSCR=-SCR(3)&-SCR(4)&-SCR(5);

    /* CHECK FOR OVERFLOW*/

    IF ROSE THEN OSF ='0'B;

    DO I=1 TO 5;
        X(I)=SCR(I);
    END;

    DO I=1 TO 2;
        Y(I)=OSTR(I);
    END;

    E01= ~X(1)&~X(2)&~Y(1)& Y(2)|~X(1)& X(2)& Y(1)&~Y(2)
        |X(1)&~X(2)& Y(1)& Y(2)| X(1)& X(2)&~Y(1)&~Y(2);

```

```

EO2=  X(1)&¬X(2)&¬Y(1)&¬Y(2)| X(1)& X(2)&¬Y(1)& Y(2)
      |¬X(1)&¬X(2)& Y(1)&¬Y(2)|¬X(1)& X(2)& Y(1)& Y(2);

EU1=  ¬X(1)&¬X(2)&¬Y(1)&¬Y(2)|¬X(1)& X(2)&¬Y(1)& Y(2)
      | X(1)&¬X(2)& Y(1)&¬Y(2)| X(1)& X(2)& Y(1)& Y(2);

EU2=  ¬X(1)&¬X(2)& Y(1)& Y(2)|¬X(1)& X(2)&¬Y(1)&¬Y(2)
      | X(1)&¬X(2)&¬Y(1)& Y(2)| X(1)& X(2)& Y(1)&¬Y(2);

OV1=  NSF |E01&CSD&(X(3)|X(4)|X(5))
      |E01&CSW&X(3)&(X(4)|X(5))
      |E01&CSH&X(3)& X(4)& X(5);

OV2=  OV1 |E02&CSD&(X(3)|X(4)|X(5))
      |E01&CSD&¬X(3)&¬X(4)&¬X(5)
      |E01&CSW&(X(3)|X(4)|X(5))
      |E01&CSH&X(3)&(X(4)|X(5))
      |E01&CSB&X(3)&X(4)&X(5);

UF1=  EU1&¬NSF&CSD
      |EU1&¬NSF&CSW&¬X(3)
      |EU1&¬NSF&CSH&¬X(3)&¬X(4)
      |EU1&¬NSF&CSB&¬X(3)&¬X(4)&¬X(5);

UF2=  UF1
      |EU2&CSD
      |EU1&¬NSF&CSW
      |EU1&¬NSF&CSH&¬X(3)
      |EU1&¬NSF&CSB&¬X(3)&¬X(4);

IF CKOFE THEN
  IF OV2 THEN
    IF OSTR(1)=T1R(1) THEN
      IF OSTR(2)=T1R(2) THEN DO;
        NSF='1'B;
        OV1='1'B;
      END;
    END;
  END;
END NPSTKIN;

```

/\*

```

ADDER5:  PROC( A5OV,      ADD5E,      C5INJ,      KBG,      NTAG,
              RSYS,      SCAG,      SCT2G,      T1R2G,      T1SCG,
              T1STG,      T2AG,      T2BG,      T2SCG);
DCL(      A5OV,      ADD5E,      C5INJ,      KBG,      NTAG,
        RSYS,      SCAG,      SCT2G,      T1R2G,      T1SCG,
        T1STG,      T2AG,      T2BG,      T2SCG) BIT(1);
DCL (OSTR(2), (K,SCR,T2R)(5), T1R(0:5)) BIT(1) EXTERNAL;
DCL (A,B)(0:5) BIT(1);
DCL I FIXED BIN;
  IF T2SCG THEN DO I=1 TO 5;
    SCR(I)=T2R(I);
  END;
  IF SCT2G THEN DO I=1 TO 5;
    T2R(I)=SCR(I);
  END;
  /* LOAD A AND B INPUTS */
  A,B='0'B;
  IF NTAG THEN BEGIN;
    /* GATE OSTR TO A */
    A(1)=OSTR(1);
    A(2)=OSTR(2);
    A(3),A(4),A(5)='0'B;
  END;
  IF SCAG THEN DO I = 1 TO 5;
    A(I)=SCR(I);
  END;

  IF T2AG THEN DO I=1 TO 5;
    A(I)=T2R(I);
  END;
  IF T2BG THEN DO I=1 TO 5;
    B(I)=T2R(I);
  END;
  IF KBG THEN DO I=1 TO 5;
    B(I)=K(I);
  END;
  /* PERFORM ADDITION */
  IF ADD5E THEN
    CALL ADD5SUB;
  /* CHECK FOR OVERFLOW */

  A5OV=ADD5E&(-C5INJ&T1R(0)|C5INJ&-T1R(0));
  /* CHECK FOR TRANSFERS INVOLVING T1R,T2R,SCR, OR THE OSTR */

  IF T1STG THEN DO I=1 TO 2;
    OSTR(I)=T1R(I);
  END;

  IF T1SCG THEN DO I =1 TO 5;
    SCR(I)=T1R(I);
  END;

  IF T1R2G THEN DO I=1 TO 5;
    T2R(I)=T1R(I);

```

```
END;  
/* RESET SYSTEM */  
IF RSYS THEN  
    A50V='0'B;
```

```
ADD5SUB: PROC;  
    DCL (I,ONE) FIXED BIN, C BIT (6), X FIXED BIN(6,0);  
    ONE=1;  
    X=0;  
    DO I = 0 TO 5;  
        IF A(I) THEN X=X+2**(5-I);  
        IF B(I) THEN X=X+2**(5-I);  
    END;  
    IF C5INJ THEN X=X+1;  
    C=BIT(X,6);  
    DO I =1 TO 6;  
        T1R(I-1)=SUBSTR(C,I,ONE);  
    END;  
END ADD5SUB;  
END ADDER5;
```

/\*

```

CONSGEN: PROC (C5INJ, CSB, CSD, CSH, CSW,
               OSCG16E, OSCG1E, OSCG2E, OSCG4E, OSCG8E,
               OSCGE, SE, X2);
DCL(C5INJ,CSB, CSD, CSH, CSW, OSCG16E,
     OSCG1E, OSCG2E, OSCG4E, OSCG8E, OSCGE,
     SE, X2) BIT(1);
DCL K(5) BIT(1) EXTERNAL;

```

/\* THE CONSTANT GENERATOR OUTPUT BUS (K) IS A 5 BIT NUMBER  
 WHICH BECOMES ONE OF THE B INPUTS TO THE 5 BIT ADDER.  
 INPUTS TO THE GENERATOR ARE THE CELL SIZE SIGNALS(CS),  
 THE ADD-SUBTRACT SIGNAL (SE), THE MULTIPLY BY 2 SIGNAL (X2),  
 THE CONTROL LINES FOR GENERATING SPECIFIC CONSTANTS (OSCG).  
 CGB IS A 5 BIT INTERMEDIATE BUS WHICH IS GATED EITHER  
 POSITIVELY OR NEGATIVELY TO K.  
 T1 AND T2 ARE DEFINED FOR CONVENIENCE TO SIMPLIFY THE  
 EQUATIONS. \*/

```

DCL CGB(5) BIT (1), (T1,T2) BIT (1);

```

```

T2=¬OSCGE&X2;
T1=¬X2;

```

```

CGB(1)=T2&CSD|OSCG16E;

```

```

CGB(2)=T1&CSD|T2&CSW|OSCG8E;

```

```

CGB(3)=T1&CSW|T2&CSH|OSCG4E;

```

```

CGB(4)=T1&CSH|T2&CSB|OSCG2E;

```

```

CGB(5)=T1&CSB|OSCG1E;

```

/\* IF NUMBER IS TO BE NEGATIVE, GATE THE CGB'S TO THE K(I)'S  
 DIRECTLY, OTHERWISE, FOR POSITIVE NUMBER, GATE ¬CGB(I)\*/

```

IF ¬SE THEN DO;
    K=CGB;
    C5INJ='0'B;
END;
ELSE DO;
    K=¬CGB;
    C5INJ='1'B;
END;

```

```

END CONSGEN;

```

/\*



## 2.3.4 Operand Stack - Logical Description

### 2.3.4.1 Operand Stack Storage

The storage used for the fast storage of the Operand Stack consists of 4 storage blocks of the type described in Section 2.2. Each block provides 8 bytes of 9 bit storage for a total of 32 bytes of fast storage.

The actual byte positions are arranged so that the first block contains bytes 0, 4, 8, ..., 28, the second block contains 1, 5, 9, ... 29, and so on.

The input to the OS storage consists of 32 select lines,  $\overline{OSi/S}$ , the Distribution Bus (DBP) from the Permuter and 4 data control lines:  $\overline{DBOS/G}$ ,  $\overline{FLOS/N}$ ,  $\overline{OSPT/G}$  and  $\overline{OSPF/G}$ . Figure 2.3.4 is an excerpt from the 01-4 drawing in the TP Logic Book. It shows a diagram of the fourth storage block in the OS fast storage and indicates the various inputs and outputs to the block.

The  $\overline{DBOS/G}$  control signal is used to gate the DB through the Input Gate Drivers into the four storage blocks. Since there is only one type of storage in these four blocks (as opposed to, say, the PR storage which also contains the Spare Buffer Register), all of the read/write enable signals to the Byte Select Drivers (BSD's) can be activated by the same signal. Thus the  $\overline{DBOS/G}$  signal is fed to 9 inverters. One of these drives the four IGD's. The other 8 each drive four read/write enable inputs to the BSD's. Thus all 32 read/write enable lines are activated more or less in unison with the IGD gate signals. Only the "Gate Both In" option is used on the IGD's.

The  $\overline{FLOS/N}$  signal is used by the OS storage blocks to inhibit writing in the flag positions.  $\overline{FLOS/N}$  drives an inverter which then feeds the In Data Inhibit signal to the Input Gates on the flag bit

TO THE REMAINING OS  
STORAGE BLOCKS

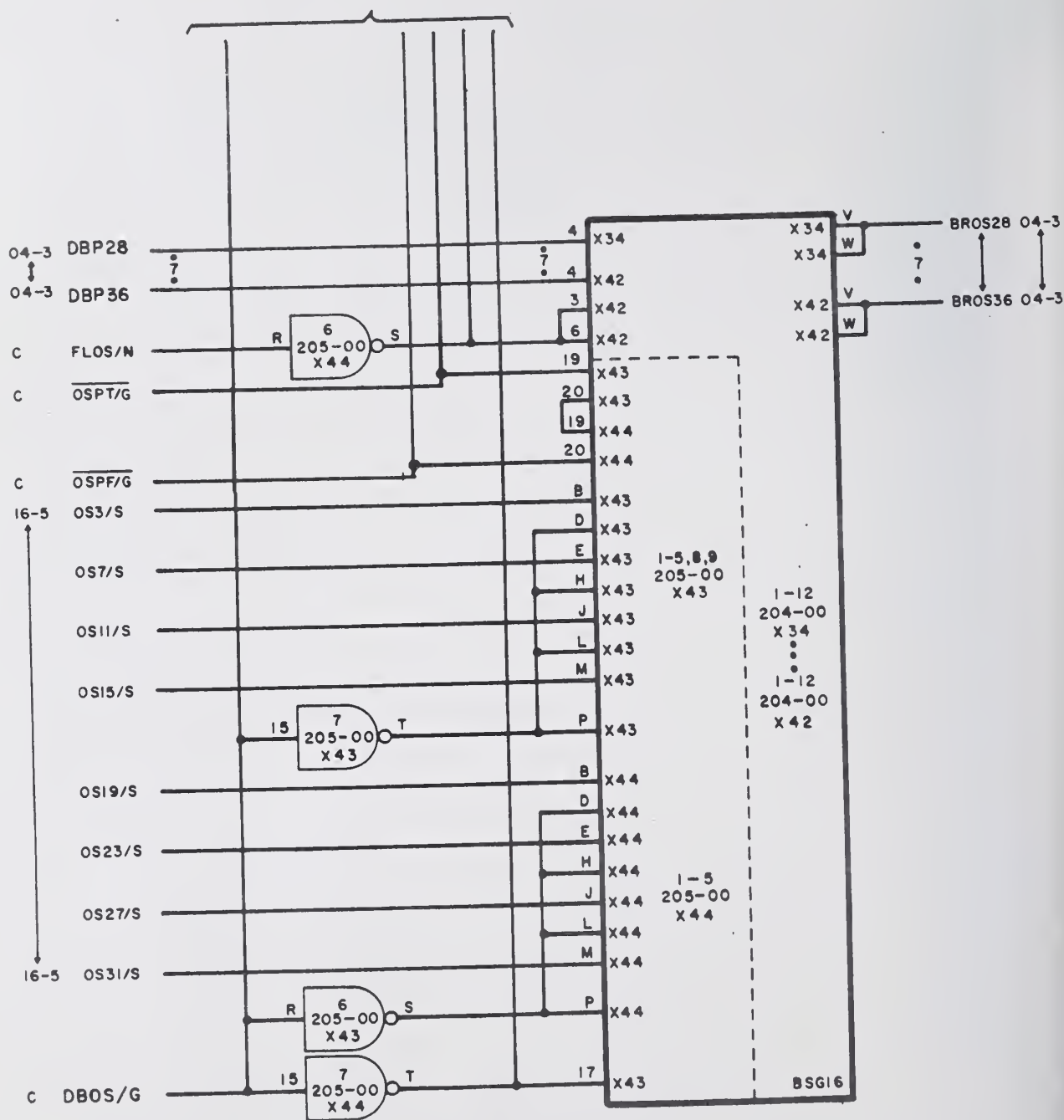


FIGURE 2.3.4

5/2/69

Section 2.3.4 - 2/3

position of every byte in the OS fast storage (see Figure 2.2.1/2).

The  $\overline{\text{OSPT/G}}$  and  $\overline{\text{OSPF/G}}$  control signals are connected to the "Gate False Out" and "Gate True Out" inputs of the Output Gate Drivers, respectively (see Figure 2.2.1/3). This seemingly reversal of the gate signals is brought about because it is necessary to dot-or the outputs of the Operand Stack Storage Blocks and thus the inverted values must be sent out. The output from the OS fast storage is the Base Register-Operand Stack Bus (BROS). When either the  $\overline{\text{OSPT/G}}$  or  $\overline{\text{OSPF/G}}$  control lines are set to "0", the true or 1's complement representation of the selected bytes in the OS fast storage are dot-ored to the BROS. Note that if both  $\overline{\text{OSPT/G}}$  and  $\overline{\text{OSPF/G}}$  are set to logical "0" at the same time, logical "0" would result in all positions.

Each block (representing 8 interleaved bytes) is gated to a separate byte of the BROS. When one of the output control lines is activated the selected bytes are gated to the bus. As explained in Section 2.3.1.3., no more than four bytes are ever gated out at one time from the OS storage. Since the blocks are permanently connected to the BROS, the OS bytes always come out in the same position on the BROS regardless of where the cell boundaries happen to be. Therefore, the Permuter is used to justify the boundaries of the BROS bytes when the information is transferred to the DB.



### 2.3.4.2 Overflow-Underflow-Logical Description

As explained in Section 2.3.1.4 the OS overflow/underflow logic can be broken up, conceptually, into several sections. First there are the equations for the position of the SCR relative to the OSTR, i.e. E01, E02, EU1, and EU2. Secondly, there are the equations for the overflow and underflow signals themselves, i.e. OV1, OV2, UF1 and UF2. Finally there is the logic involved in setting and using the OS Full flip-flop, OSF. All three sections of logic are shown in Drawing 16-6 of the TP Logic Book and will now be explained.

The equations for E01, E02, EU1 and EU2 as given previously in Section 2.3.1.4 are fairly straightforward to implement. The diode matrix board, 236-01, which is shown in Figure 2.3.4.2/1 as a logical circuit and in Figure 2.3.4.2/2 as a schematic, is used to calculate the equations. This board is simply a four-bit decoder which decodes all possible combinations of the two high order SCR and OSTR bits. The inputs to the diode matrix are driven by 228-07 circuits. The 16 individual outputs of the diode matrix are sent to 228-05 circuits, the outputs of which are dot-or'ed to generate the four signals according to the equations in Section 2.3.1.4. These signals are inverted by 4 228-07 circuits for use in the OV1, OV2, UF1 and UF2 equations.

The equations for OV1, OV2, UF1, and UF2 are considerably more difficult to implement. As shown in Section 2.3.1.4 they are as follows:

$$\begin{aligned}\emptyset V1 &= \emptyset SF \vee E\emptyset 1 \cdot CSD \cdot (X_3 \vee X_4 \vee X_5) \\ &\vee E\emptyset 1 \cdot CSW \cdot X_3 \cdot (X_4 \vee X_5) \vee E\emptyset 1 \cdot CSH \cdot X_3 \cdot X_4 \cdot X_5 \\ \emptyset V2 &= \emptyset V1 \vee E\emptyset 2 \cdot CSD \cdot (X_3 \vee X_4 \vee X_5) \\ &\vee E\emptyset 1 \cdot CSD \vee E\emptyset 1 \cdot CSW \cdot (X_3 \vee X_4 \vee X_5) \\ &\vee E\emptyset 1 \cdot CSH \cdot X_3 \cdot (X_4 \vee X_5) \vee E\emptyset 1 \cdot CSB \cdot X_3 \cdot X_4 \cdot X_5\end{aligned}$$

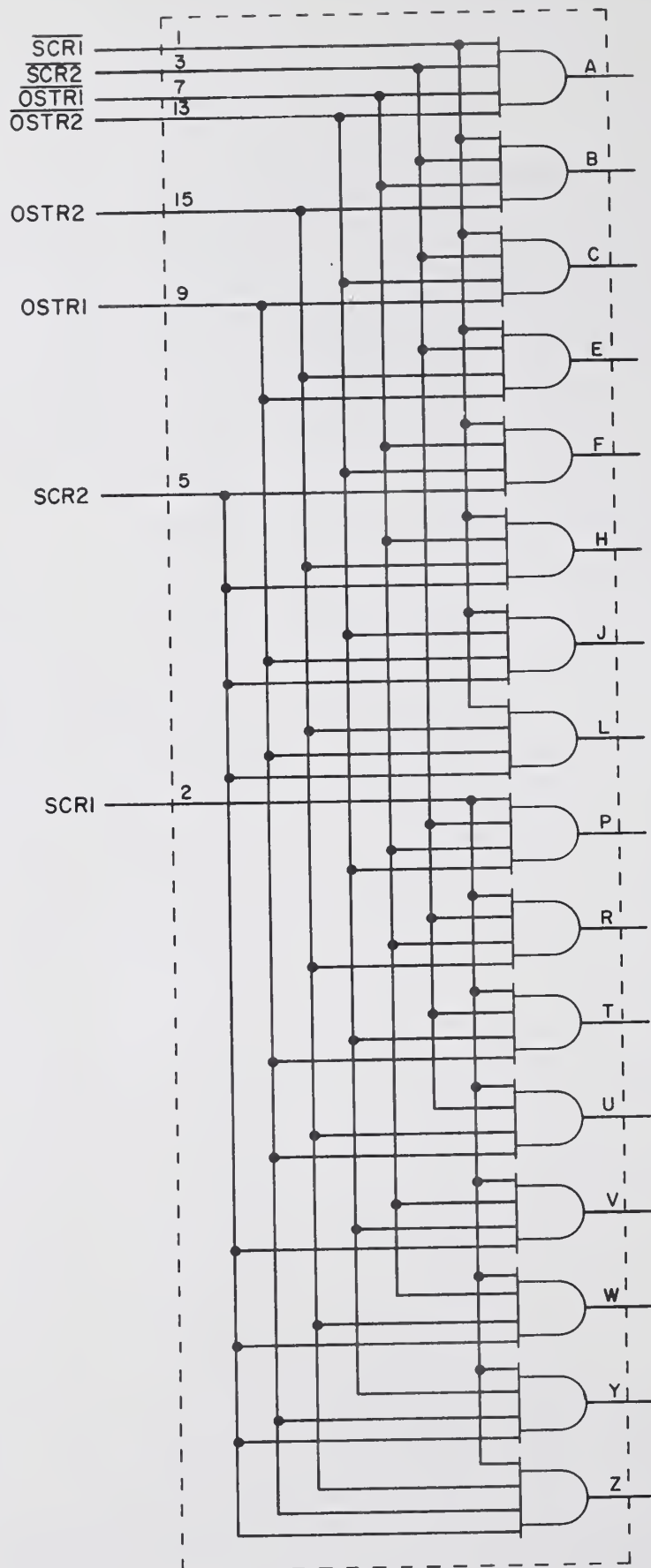


Figure 2.3.4.2/1 - SCR-STR Decoder for  
E01, E02, EU1, EU2 Signals Logic Representation

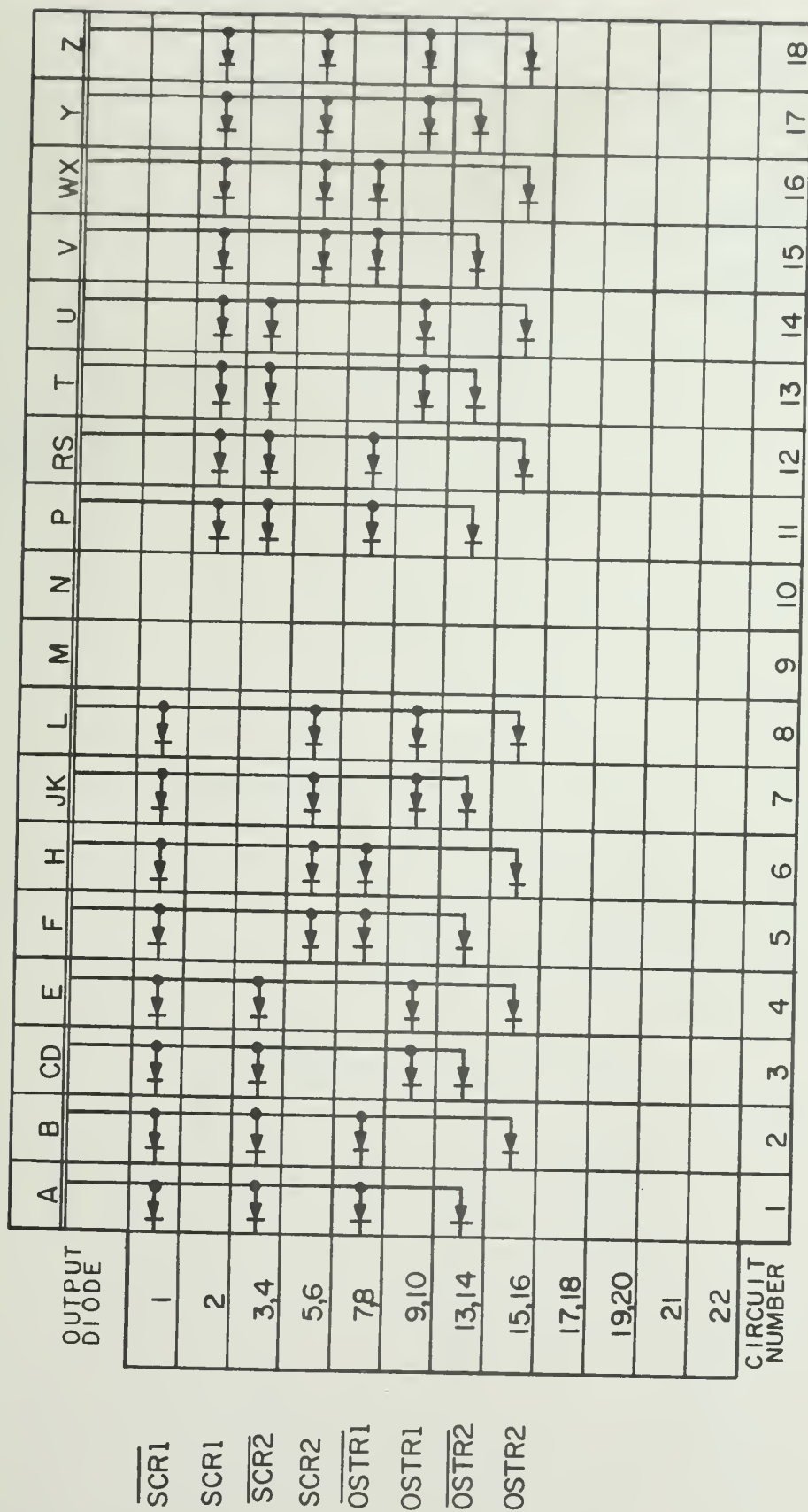


Figure 2.3.4.2/2 - SCR-OSTR Decoder for E01, E02, EU1, EU2 Signals  
Diode Matrix Representation



$$UF1 = EU1 \cdot \overline{OSF} \cdot CSD \quad \vee \quad EU1 \cdot \overline{OSF} \cdot CSW \cdot \overline{X}_3 \\ \vee \quad EU1 \cdot \overline{OSF} \cdot CSH \cdot \overline{X}_3 \cdot \overline{X}_4 \quad \vee \quad EU1 \cdot \overline{OSF} \cdot CSB \cdot \overline{X}_3 \cdot \overline{X}_4 \cdot \overline{X}_5$$

$$UF2 = UF1 \\ \vee \quad EU1 \cdot \overline{OSF} \cdot CSW \quad \vee \quad EU1 \cdot \overline{OSF} \cdot CSH \cdot \overline{X}_3 \\ \vee \quad EU1 \cdot \overline{OSF} \cdot CSB \cdot \overline{X}_3 \cdot \overline{X}_4$$

where, as before,  $X_i$  represents  $SCR_i$ .

When implemented by the logic in Drawing 16-6, however, these equations look considerably different. For OV1 we have:

$$OV1 = OSF \vee (E\emptyset 1 \cdot CSD \cdot (X_3 \vee X_4 \vee X_5) \\ \vee \quad E\emptyset 1 \cdot CSW \cdot (X_3 \cdot (X_4 \vee X_5)) \\ \vee \quad E\emptyset 1 \cdot CSH \cdot (X_3 \cdot X_4 \cdot X_5) )$$

Note that OV1 consists essentially of two terms which are inputs to a 214-03 driver going to the control logic. The first term comes directly from the output of the OSF flip-flop. The second term is produced by dot-oring 3 NAND circuits together. Each of these NAND's has as inputs the E $\emptyset$ 1 signal, a cell size signal, and some function of the 3 low-order bits of the SCR.

The equation for OV2 as implemented by the logic can be written as follows:

$$OV2 = OSF \vee (E\emptyset 1 \cdot CSD \cdot (X_3 \vee X_4 \vee X_5) \\ \vee \quad E\emptyset 1 \cdot CSW \cdot (X_3 \cdot (X_4 \vee X_5)) \\ \vee \quad E\emptyset 1 \cdot CSH \cdot (X_3 \cdot X_4 \cdot X_5) ) \\ \vee \quad (E\emptyset 2 \cdot CSD \cdot (X_3 \vee X_4 \vee X_5) \\ \vee \quad E\emptyset 1 \cdot CSD \\ \vee \quad E\emptyset 1 \cdot CSW \cdot (X_3 \vee X_4 \vee X_5) \\ \vee \quad E\emptyset 1 \cdot CSH \cdot (X_3 \cdot (X_4 \vee X_5)) \\ \vee \quad E\emptyset 1 \cdot CSB \cdot (X_3 \cdot X_4 \cdot X_5) )$$



This equation consists of three terms all of which are inputs to a 214-03 driver going to the control logic. The first two terms are identical to the ones used for OV1. These replace the OV1 term in the original OV2 equation and eliminate the need to invert the OV1 output and use it as an input to OV2, thus saving 2 collector delays. The third term has the same format as the last term in OV1. It consists of 5 NAND circuits dot-or'ed together. Each NAND circuit has as inputs either EØ1 or EØ2, a cell size signal, and a function of the low-order 3 bits of the SCR. Note that three of these SCR terms are functions which were also used in OV1, except that in that case they were associated with different cell size signals.

The logic implementation for UF1 and UF2 are somewhat confusing. This is due, in part because they are partially implemented on the 16-6 drawing using DTL logic and partially implemented in the main control logic using IC's. Because of several changes which were necessary after the initial wiring of this logic, the structure is somewhat haphazard. The actual implementation equations for UF1 and UF2 are as follows:

$$\begin{aligned} UF1 = & (\overline{OSF} \cdot (EU1 \cdot CSD \\ & \vee EU1 \cdot CSW \cdot \overline{X_3} \\ & \vee EU1 \cdot CSH \cdot (\overline{X_3} \cdot \overline{X_4}) \\ & \vee EU1 \cdot CSB \cdot (\overline{X_3} \cdot \overline{X_2} \cdot \overline{X_1}))) \end{aligned}$$

$$\begin{aligned} UF2 = & (EU2 \cdot CSD) \\ & \vee (\overline{OSF} \cdot (EU1 \cdot CSW \\ & \vee EU1 \cdot CSH \cdot \overline{X_3} \\ & \vee EU1 \cdot CSB \cdot \overline{X_3} \cdot \overline{X_4})) \\ & \vee (\overline{OSF} \cdot (EU1 \cdot CSD \\ & \vee EU1 \cdot CSW \cdot \overline{X_3} \\ & \vee EU1 \cdot CSH \cdot (\overline{X_3} \cdot \overline{X_4}) \\ & \vee EU1 \cdot CSB \cdot (\overline{X_3} \cdot \overline{X_4} \cdot \overline{X_5}))) \end{aligned}$$

Note that UF1 consists of one term and UF2 consists of three terms, the last of which is identical to the term for UF1. This is done for the same reasons as before with OV2 and OV1.

In the case of UF1 and UF2, the final signals are formed by IC NAND circuits in the control logic section of the TP. Thus the single term for UF1 (which is also the last term of UF2) is given the name  $\overline{UF12C}$  and is formed by a 214-03 circuit using two inputs. The first input is  $\overline{OSF}$  and the second is the inverted output of a dot-or connection between four NAND circuits. Each of these four NAND's has as an input the EU1 signal, a cell size signal and some function of the 3 low-order bits of the SCR.

In addition to the  $\overline{UF12C}$  signal, the UF2 signal has two other terms which have been given the names  $\overline{UF2A}$  and  $\overline{UF2B}$ . The first term,  $\overline{UF2A}$ , consists of the output of one 214-03 NAND which forms the term  $\overline{EU2 \cdot CSD}$ . The second term also comes from a 214-03 circuit but uses two inputs. The first is  $\overline{OSF}$ . The second is the inverted output of a dot-or connection between 4 NAND circuits. Each of these NAND circuits has as an input the EU1 signal, a cell size signal, and some function of the 3 low-order bits of the SCR. The 2 NAND circuits which form UF1 and UF2 in the TP Control section are shown in Figure 2.3.4.2/3.

The final section of logic in the OS overflow/underflow logic is the OSF flip-flop itself. As discussed in Section 2.3.1.4, the setting and resetting of the OSF flip-flop is determined by the Main Control Logic, specifically the OS ENTRY and SCR MOD sequences.

The OSF flip-flop will be set to "1" whenever the CKOF/E signal goes to "1" and at the same time the two high order bits of the SCR are equal to the OSTR, the three low-order bits of the SCR are "0", and the EU1 signal is on. In this situation the SCR is moving to the OSTR boundary from the "full" side of the hardware registers.

OSF is reset by Main Control using the  $\overline{ROSF}$  control line.

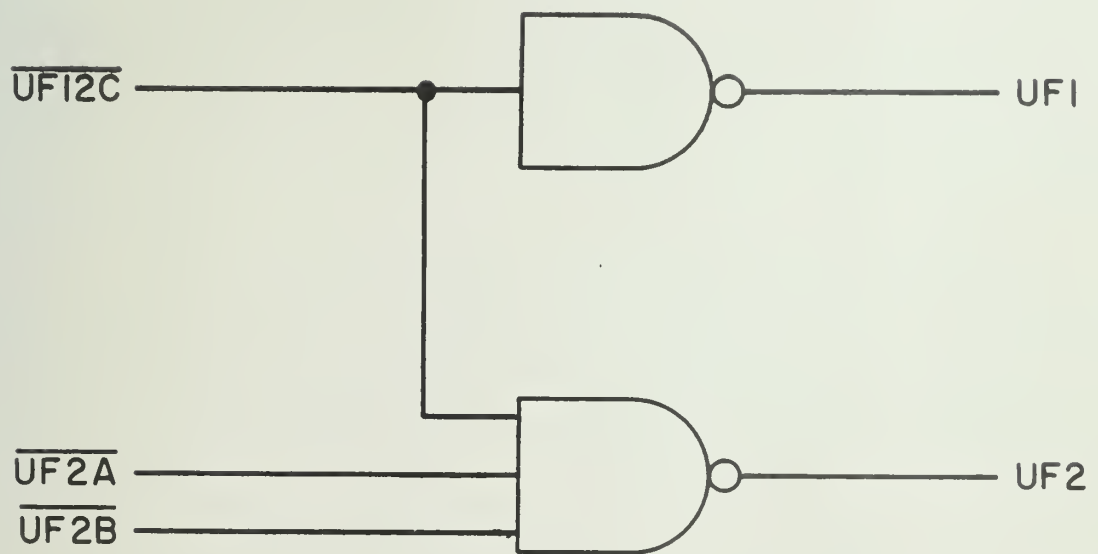


Figure 2.3.4.2/3 - UF1 and UF2 IC Logic Implementation



#### 2.3.4.3 5-Bit Adder-Logical Description

The 5-Bit Adder is a full carry lookahead adder generally used for adding or subtracting constants from various operand stack control registers. In basic construction it is somewhat similar to the 32-Bit Adder (Section 2.7) except, of course, it is much simpler since it has much fewer bit positions. The two 5 bit input busses are the A bus and the B bus. The A bus is usually loaded with one of the Operand Stack Registers and the B bus is usually loaded with a  $\pm$  constant from the Constant Generator. The results of the addition are stored in temporary Register T1R.

At each input bit position of the 5-bit Adder there are 4 NAND gates. These NAND's are all activated by the ADD5/E signal and produce for each bit position, i, a generate signal, GENi, and a transfer signal, TRANi.

When a generate signal is on, it indicates that for the particular bit position, i, a carry will be produced out of it independently of whether or not a carry was made into the position. This can be expressed as a function of the two input data bits for that position by the equation:

$$GENi = A_i \cdot B_i$$

When a transfer signal is on, it indicates that for the particular position, i, a carry will be produced out of it only if a carry is made into it. This can be expressed as a function of the two input data bits for that position by the equation:

$$TRANi = A_i \cdot \overline{B_i} \vee \overline{A_i} \cdot B_i$$

Note that if a GENi signal is on for a given bit position the TRANi signal for that bit position cannot be on and if the TRANi signal is on, the GENi signal cannot be on. However, it is possible for both signals to be off at the same time.

As can be seen in Drawing 16-3 of the TP Logic Book, these two signals are actually each generated by "dot-oring" 2 NAND circuits together. Figure 2.3.4.3/1 shows the GENi and TRANi logic for one bit

position of the 5-bit Adder. The so called "dot-ors" must really be considered as AND circuits in the sense that if either input is forced to zero, the output will be forced to zero. Thus we have:

$$\begin{aligned} \text{GEN}_i &= \overline{(\overline{A_i})} \cdot \overline{(\overline{B_i})} \\ &= A_i \cdot B_i \end{aligned}$$

$$\begin{aligned} \text{TRAN}_i &= \overline{(A_i \cdot B_i)} \cdot \overline{(\overline{A_i} \cdot \overline{B_i})} \\ &= \overline{(A_i \cdot B_i)} \vee (\overline{A_i} \cdot \overline{B_i}) \\ &= (A_i \cdot \overline{B_i}) \vee (\overline{A_i} \cdot B_i) \end{aligned}$$

Once these two signals have been generated, they can be used to calculate the carries into every bit position of the adder by using the following equations:

$$C5IN5 = C5INJ$$

$$C5IN4 = G5 \vee T5 \cdot C5INJ$$

$$C5IN3 = G4 \vee T4 \cdot G5 \vee T4 \cdot T5 \cdot C5INJ$$

$$C5IN2 = G3 \vee T3 \cdot G4 \vee T3 \cdot T4 \cdot G5 \vee T3 \cdot T4 \cdot T5 \cdot C5INJ$$

$$\begin{aligned} C5IN1 &= G2 \vee T2 \cdot G3 \vee T2 \cdot T3 \cdot G4 \vee T2 \cdot T3 \cdot T4 \cdot G5 \\ &= \vee T2 \cdot T3 \cdot T4 \cdot T5 \cdot C5INJ \end{aligned}$$

$$\begin{aligned} C5\emptyset V &= G1 \vee T1 \cdot G2 \vee T1 \cdot T2 \cdot G3 \vee T1 \cdot T2 \cdot T3 \cdot G4 \vee \\ &= \vee T1 \cdot T2 \cdot T3 \cdot T4 \cdot G5 \vee T1 \cdot T2 \cdot T3 \cdot T4 \cdot T5 \cdot C5INJ \end{aligned}$$

where  $G_i$  and  $T_i$  represent  $\text{GEN}_i$  and  $\text{TRAN}_i$  respectively,  $C5IN_i$  is the input carry into bit position  $i$ ,  $C5INJ$  is an injected carry into the low order bit (used in subtraction to convert a 1's complement constant to 2's complement), and the adder bit positions are numbered beginning with 1 at the highest order position.

The last 4 equations are implemented using the 236-00 Lookahead Carry Generator diode matrix board shown in Figure 2.3.4.3/2. The two initial ones are implemented using NAND circuits. The outputs are sent to the 5 output positions of the 5-bit Adder and the overflow indicator. The Adder outputs are determined by the equation:

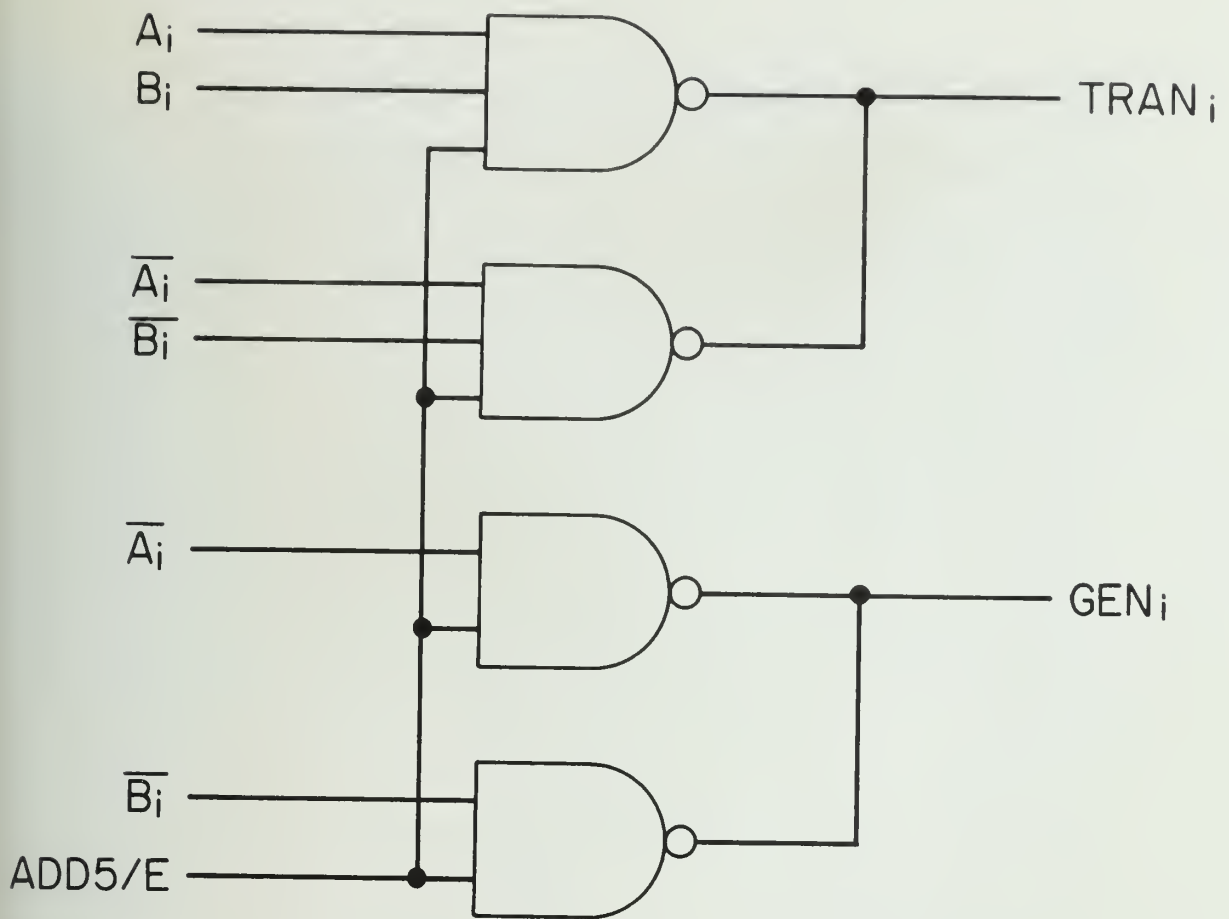


Figure 2.3.4.3/1

Generation of  $GEN_i$  and  $TRAN_i$  in the 5-Bit Adder



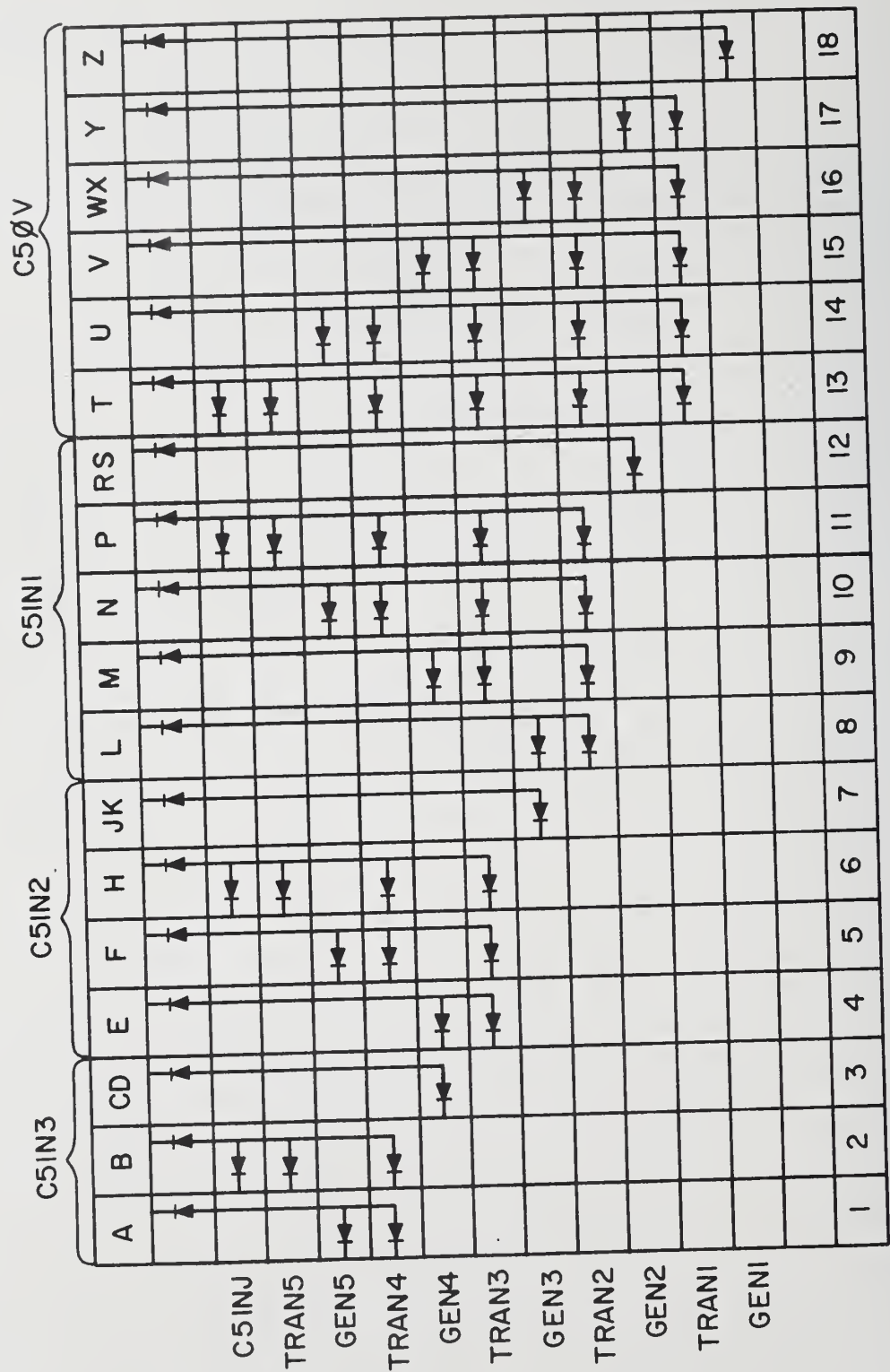


Figure 2.3.4.3/2

5-Bit Adder Lookahead Carry Generator - Diode Matrix - 236-00



$$TlRi = TRANi \cdot C5INi \vee \overline{TRANi} \cdot C5INi$$

The overflow bit is determined by:

$$A5\emptyset V = \overline{C5INJ} \cdot C5\emptyset V \vee C5INJ \cdot \overline{C5\emptyset V}$$

This merely means that if an addition sum ( $C5INJ = 0$ ) is greater than 31 or if a subtraction difference ( $C5INJ = 1$ ) is less than 0, the  $A5\emptyset V$  flip-flop will be set. This does not necessarily indicate a fatal error. As is explained in Sections 2.3.1.4 and 4.5.1.4, this condition is used to indicate stack wraparound.

Note that in the logic in Drawing 16-3 of the TP Logic Book the inputs to the temporary register  $TlR$  and the 5-bit Adder Overflow Flip-Flop,  $A5\emptyset V$ , are enabled by the  $ADD51E$  signal.



## 2.4 The Pointer Registers

There are 15 Pointer Registers (PR's) in the Taxicrinic Processor. These registers are in some ways similar to the index registers of more conventional machines, although they are much more powerful. As with conventional index registers, the PR's may be incremented, modified and tested. Thus they can be used in any of the ways that an index register might be used, except that these operations usually appear quite different in the TP because of the unusual machine organization.

Primarily, however, the PR's are a means of addressing storage. All storage accessing is performed "through" the PR's, by indicating the name of the PR which contains the address desired instead of requesting the actual address (the contents of the PR can be modified as noted above). The indicated PR will contain a segment name and an address in that segment relative to the segment base address. By loading the PR's beforehand, the programmer can refer to any address in memory by using only a 4-bit tag to indicate the name of the PR to be chosen. (For a more detailed description of memory accessing, see Section 4.1).

In order to be able to choose a particular PR, each PR has associated with it a 4-bit Name Register (NR). Each NR holds the current "name" assigned to its associated PR. These names are unique but not permanent, i.e., the name of the PR (which is a number for 0 to 14) can be changed, but only one PR has any given name at any given time. The process whereby the PR names are changed is called "name permutation" and is described in detail in Section 4.2.4.3.

Since the PR's are the only means by which memory may be accessed, the address of a variable must be assigned to its particular PR prior to its first call. Provisions are provided in the instruction set to initialize all designated PR's either one-at-a-time or by multiple assignment (using an imprimitive instruction, see Section 4.2.4).

#### 2.4.1 Pointer Register Formats

There are three formats for the pointer registers. The "normal" format is used in connection with the pointer stacks as shown in Figure 2.4.1.1 of Section 2.4.1.1.

The "list processing format" is used with the list processing instructions and does not utilize pointer stacking. Instead the pointer is divided into two half-word link fields which are identical to the contents in the cell currently being "looked at" by the pointer register.

The third format for pointer registers is the "available space" format. This format is used whenever a PR is utilized to control the assignment of list processing cells. In particular PR14, which controls the storage of the pointer registers in the pointer stacks, is in this format.



#### 2.4.1.1 Pointer Stack Format

Reflecting the nested nature of operand calls, pointer stacks are provided into which the current value of a PR can be pushed for safekeeping. When this value is again required, the stack can be popped and the value reloaded into the PR. The pointer stacks are maintained in main storage under the control of PR14, which is in the "available space" format (see Section 2.4.1.3).

The pointer stacks themselves are ordinary unidirectional linked lists. The pointer stack cells take up two words of storage, but only the first 6 bytes are currently being used. As shown in Figure 2.4.1.1, the first two bytes are used to hold the 16 bit address of the previous entry in the stack, while the next two bytes contain the relative address within the segment. The final 2 bytes contain the segment name.

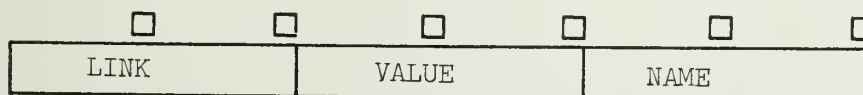


Figure 2.4.1.1 Pointer Stack Cell

The physical Pointer Registers in the TP actually consist of 2 registers, a 4 byte register containing the link and value fields and a 2 byte register containing the name field. This configuration of the pointer register is referred to as the "normal" or "pointer stack" format. It should be noted that all flag bits are always pushed into the pointer stacks along with the link, value and name fields.





#### 2.4.1.2 List Processing Format

The List Processing format for the pointer registers differs from the normal PR format. As shown in Figure 2.4.1.2 it consists of a left and right link instead of a link and value field. The name field remains the same and indicates the name of the segment being used as the base for the two links. There are no stacks connected with PR's in this format.

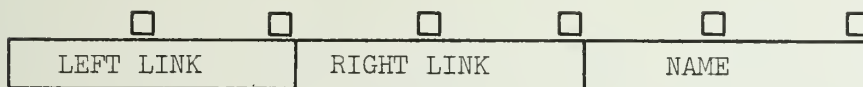


Figure 2.4.1.2 List Processing Format

In the list processing format, a PR can be considered to be a "bug" which moves along a list structure in core. At any given time the PR contains an image of the first four bytes of the cell currently being "looked at". These first four bytes will usually be interpreted as two separate pointers. It should be noted that the PR does not contain a pointer to the cell being looked at. It only contains the contents of the first 4 bytes of that cell. In the conventional method of using the Illiac III list processing instructions (specifically SL and SR), the address of the current cell will be available at the top of the OS.

In use the PR can be used to sequence through a list structure. At any given point, cells may be added or deleted to the "right" or "left" of the current cell. Links in the structure may also be changed by using the ASSIGN statement to load the contents of some PR into a cell in the structure. Instructions such as POP and PUSH can also be used to load and unload links from the OS.



### 2.4.1.3 Available Space Format

The Available Space format is used whenever a PR is utilized to control the assignment of list processing cells. PR#14, which controls the storage of pointer registers in the pointer stacks, is always in this format. As shown in Figure 2.4.1.3/1 the pointer register in the available space format consists of a link field and a count field. The name field retains the same interpretation as for the normal and list processing formats, and indicates the name of the segment being used as the base for the link and count relative addresses.

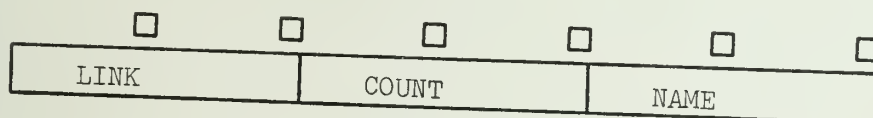


Figure 2.4.1.3/1 Available Space Format

The basic idea behind this format type is to set aside an available space file in memory to be used in the construction and destruction of list structures. Automatic allocation features of this file — when using the "available space" format for the associated PR — allow the programmer to obtain empty cells and to discard unneeded cells.

An available space file may be divided up into two parts: a section of list storage occupying the low address end of the file and a section of consecutive storage located in the high address end of the file. The list storage part consists of a sequence of cells; here the first 2 bytes of each cell contain the link to the next cell while the rest of the cell may eventually contain data. The size of these two areas changes dynamically as the file is used. At the very beginning of use the file is completely made up of consecutive storage, but as more storage is needed for the linked lists,

this area is absorbed into the list storage area. Figure 2.4.1.3/2 gives an example of the appearance of the file after it has been in use awhile.

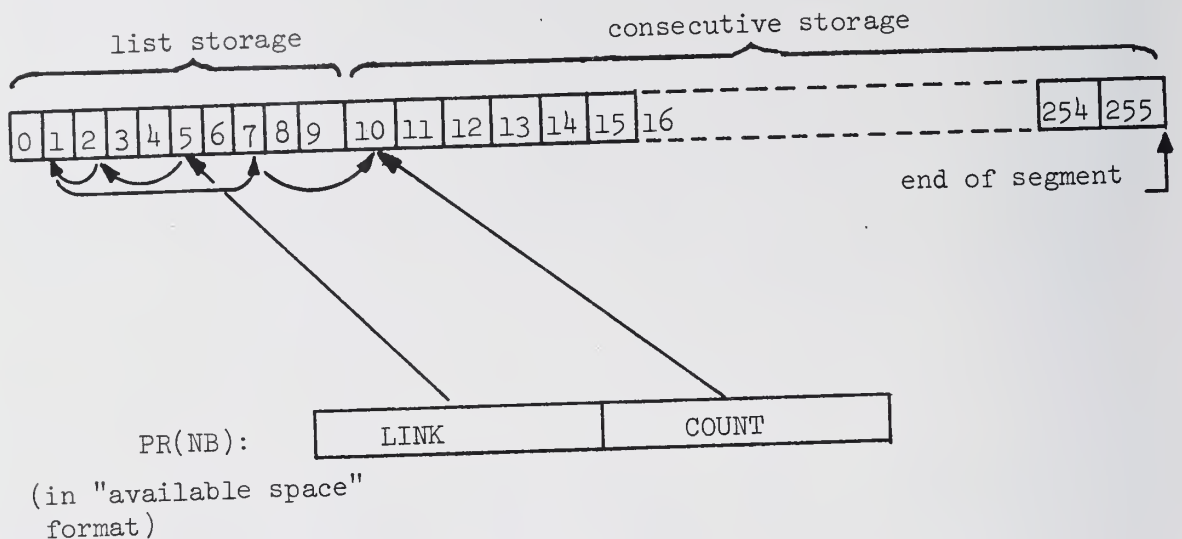


Figure 2.4.1.3/2 Available Space File

Referring to Figure 2.4.1.3/2 whenever a list cell presently assigned to a user is no longer needed by the user (i.e. it is "popped" off the user's list), it is placed on a special list (in the list storage area) called the "free list". This list contains all the cells in the list storage area which have been "freed" to be used as storage for other lists. The Pointer Register (in "available space" format) contains a link to the top cell on this list so that all access to the free list is made through the available space PR. In the example of Figure 2.4.1.3/2 the free list link would equal the address of cell 5. Cells 5, 2, 1 and 7 are on the "free list".

Whenever data needs to be stored on one of the user lists in the list storage area, the available space PR is modified so that it is

linked to the second available free list cell, and the top cell on the free list can then be used to store the new data and its link. For example in Figure 2.4.1.3/2 if a cell were needed by one of the user lists, the free list link of the available space PR would be loaded with the link portion of cell 5, i.e. the address of cell 2 in this case. Then cell 5 could be used by the list which needed a new cell.

Only cells of one size may be placed in any given available space file. The size of the cells in a given file is determined by a number initially placed in the leftmost halfword of the zeroth cell in the file. (When using PR14, the cell size is known to be 4, so this latter procedure is ignored.) In general any even integer  $\leq 256$  may be used for the cell size (for other than PR14). This zeroth cell is never distributed since its address would conflict with the symbol for the end of a list (i.e. link field of zero). Accordingly an available space pointer is normally initiated with  $LINK = COUNT = \text{cell size} > 0$ .

But what happens if the free list is empty? To solve this problem the last cell in the free list (#7 in the illustration) has its link pointing to the lowest addressed cell in consecutive storage. The second two bytes of the available space PR (called the COUNT) also contains this address, so that whenever the LINK equals the COUNT, the free list has been exhausted. When this happens and a request for another cell is received, the first cell of consecutive storage is used and both the count and link are incremented by the number of bytes in the cell. (However, if the assigned cell crosses or exceeds the segment boundary, an Available Space Empty interrupt is generated. In this latter case the count and link are not incremented, and therefore reflect the arrangement of the available space at the time of the illicit memory access.)

As an example, in Figure 2.4.1.3/2 if the free list became exhausted, the LINK and COUNT would both be equal to the address of cell 10. If a request for a cell were then made, cell 10 would be given out and both LINK and COUNT would be incremented by the value stored in the first halfword of the zeroth cell. LINK and COUNT would then point to the address of cell 11.

The count receives its name from the fact that it is the number of cells (times the cell size) in the list storage area. When the count equals the cell size, all storage is consecutive. (The zeroth cell is dedicated to the cell size.) When the high order byte of the count equals the address bounds of the segment all storage in the segment is of the list type.

One precaution should be noted. Since a PR in available space mode does not have a pointer stack of its own, it cannot be pushed into one. As a result a programmer must be extremely careful to avoid pushing an available space format PR or drastic confusion may result. In general there is nothing in the hardware to prevent this. However in the case of PR#14, an interrupt will be initiated.



## 2.4.2 Pointer Registers - Functional Description

### 2.4.2.1 Pointer Register Storage

The Pointer Register contents, consisting of a link field, a value field, and a segment name field (or link, count and segment name fields in the available space format), are stored in two separate areas of the TP mainframe. This is mainly because the use of segment names was not developed until after the rest of the PR logic had been designed.

The first two PR fields are stored in storage blocks whose design is explained in Section 2.2. Each PR has 4 storage bytes consisting of 8 data bits and 1 flag bit each. This PR storage is divided into two groups, PRO through PR7 and PR8 through PR14. Also contained in the second group is the Spare Buffer Register (SBR), which is physically exactly like the first 4 bytes of a PR. It is not treated as a PR, however. Instead it is intended to be an additional storage register to be used by the TP control unit. It is not accessible to the programmer.

As shown in diagrams 01-0, 01-2 and 01-3 of the TP Logic Book, each group of PR storage is made up of 4 of the 9 bit/byte storage blocks described in Section 2.2. Each of these blocks represents 1 byte position of a PR. Since each 204-00 circuit board contains 8 fast register flip-flops, this allows 8 registers in each group.

The data input to the PR storage blocks consists of the Distribution Bus from the Permuter (DBP). The selection lines from the Pointer Register Selection Logic operate the byte select lines to the byte select drivers (BSD) in the blocks. For writing data into the PR's only the "Gate Both In" option is used by the Input Gate Drivers (see Section 2.2.1). This signal is controlled by DBPR/G and gates the Distribution Bus into the PR storage flip-flops selected by the PR Selection Logic. The DBPR/G signal is common to all 15 Pointer Registers.

The other control inputs to the PR storage blocks are WRPRL/E, the write PR link field enable signal, and WRPRV/E, the write value field enable signal. These signals are inverted and then become the Read/Write input to the BSD's as described in Section 2.2.1. The WRPRL/E and WRPRV/E are common to all 15 Pointer Registers in the two storage groups.

The data outputs of the two PR storage groups are dot or'ed together to give a single output line in each position, called PRBAi. This presents a minor problem since if both output gates in a given position are activated with a read/write input of "0", the group which does not have a select signal activated will have as its output all zeros and this will "erase" the output of the group desired. Therefore a means to gate out only that group which has a word selected in it must be provided.

This means is provided by the PR7G/S and PR8G/S signals. They arise in the PR Selection Logic. If one of the PR's 0 to 7 is selected, the PR7G/S is "1", enabling the proper group to be gated out. PR8G/S is always the inverse of PR7G/S, so if one group is not in a "selected" state, the other is. This selection need not be made for input gating since the PR actually written into is controlled by the PR select input.

Only the "Gate False Out" signal is used in the PR storage blocks, since an inverted signal is desired at the output. The method of choosing the select lines is explained in Section 2.4.2.3.

The final PR field, the segment name, is stored in a special block of IC storage. This storage block consists of 16 registers, each 16 bits long and is made up of SN7475N quadruple bistable latch chips. The storage itself is organized into subgroups of 8 registers x 2 bits, as shown in Figure 2.4.2.1/1.

The data input lines come from the IC driver extension of the DB. The output of the segment name storage is the PR Segment Name Bus, PRSNB, which leads to the association logic and can also be gated to the PRB input to the permuter.



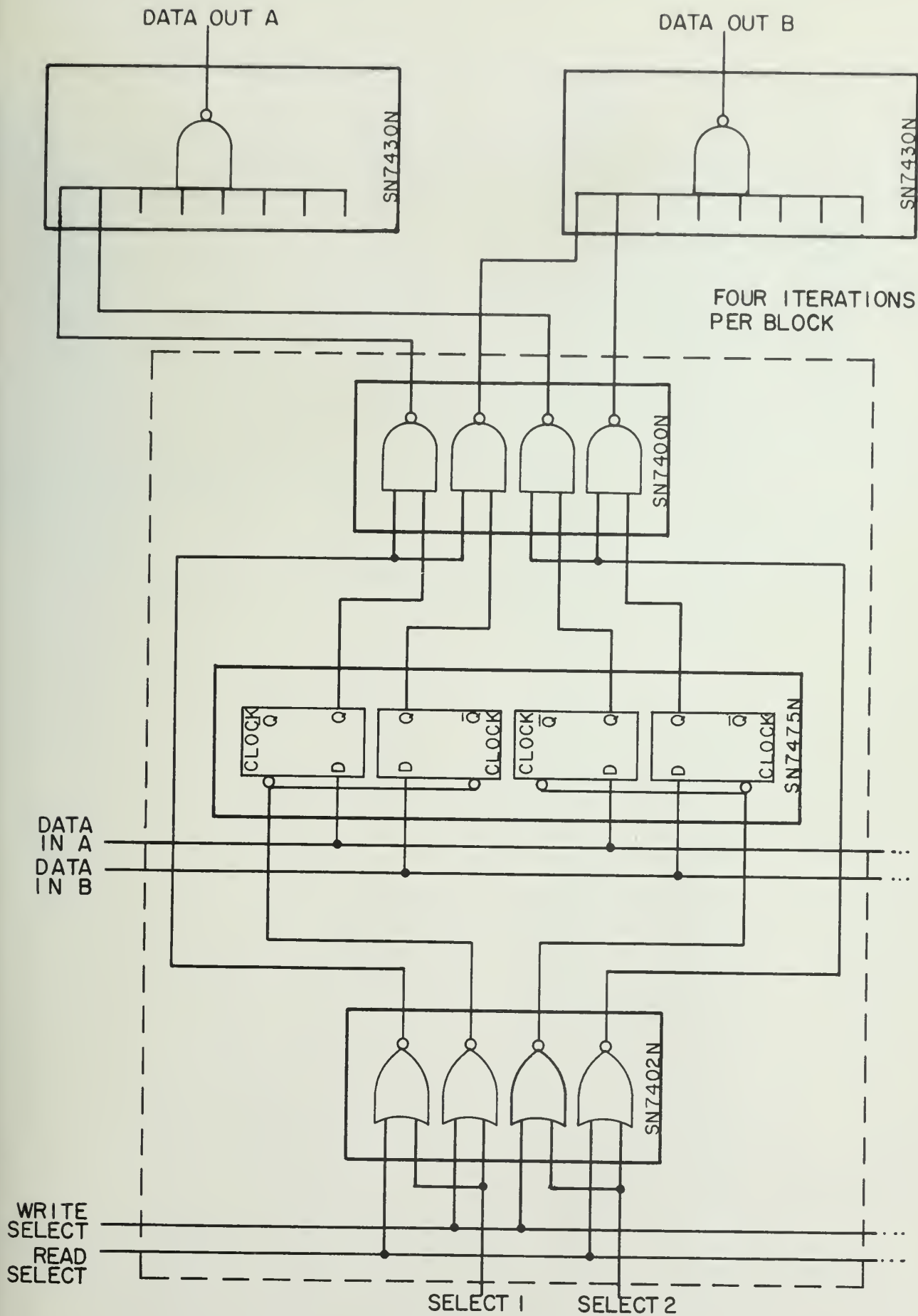


Figure 2.4.2.1/1-PR Segment Name Storage Registers  
(Block Includes 8 Registers x 2 Bits  
and Read/Write Logic)

To select the desired register, the logic shown in Figure 2.4.2.1/2 is used. When the PR Selection Logic determines the PR which which is to be accessed (see Section 2.4.2.3), the proper selection line,  $\overline{\text{PRi/S}}$ , will be activated and can be gated into the PR Segment Name Register Selector Logic by activating  $\overline{\text{SNS/G}}$ . Then in order to read out the selected Segment Name,  $\overline{\text{SNRD/E}}$  must be set to 0. As shown in Figures 2.4.2.1/1 and 2.4.2.1/2 this will activate one of the  $\text{PSNRi/S}$  lines which in turn will cause the selected PR Segment Name Register to appear on the PRSNB. A write can be performed by activating  $\overline{\text{SNWT/E}}$ .

Note that the selector is not attached to PR Segment Name Register #15. Since there is no PR#15, this segment name register is used as temporary storage by the Memory Sequence. It has independent read and write control lines so that it can be operated without affecting the selector logic.

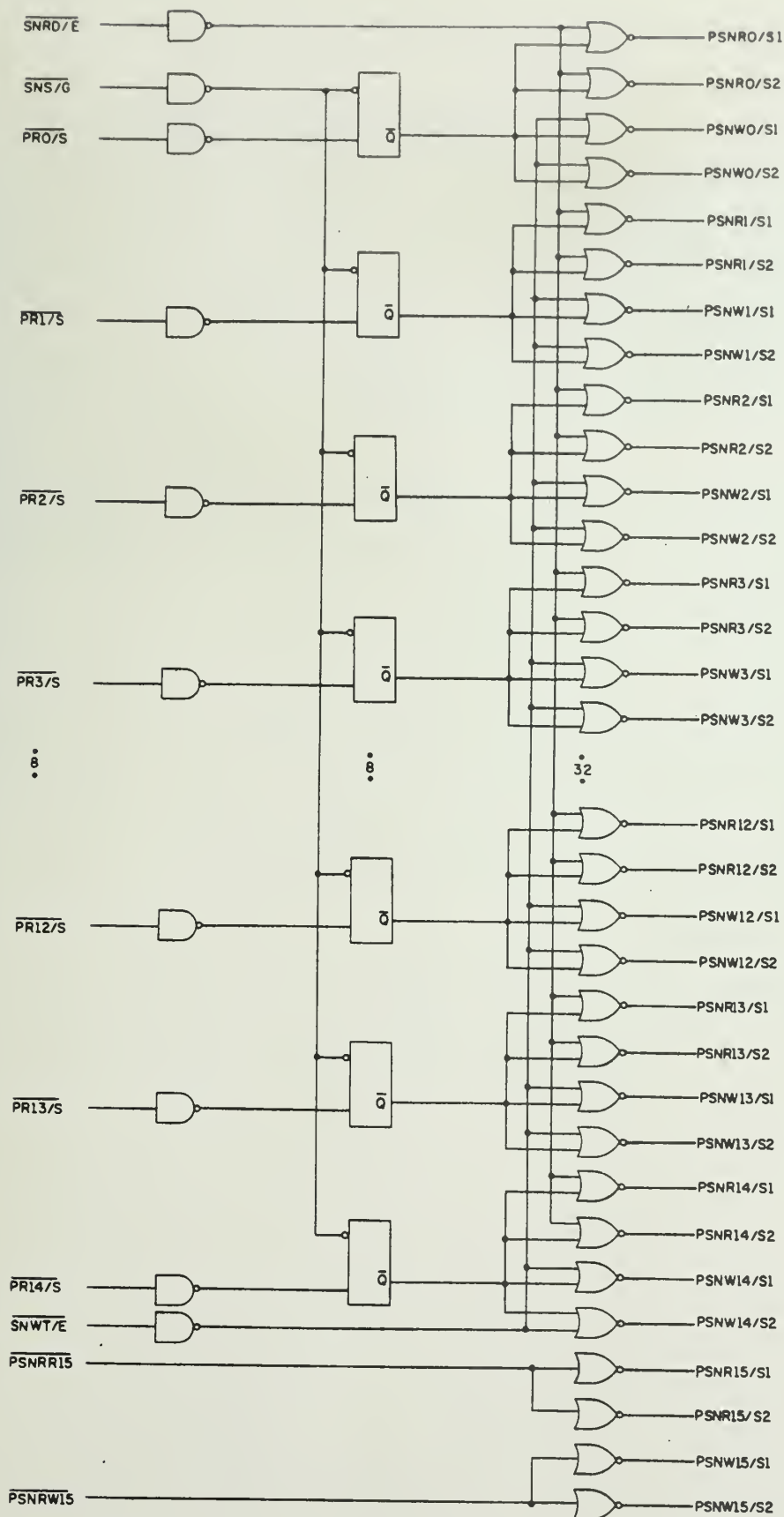


Figure 2.4.2.1/2 - PR Segment Name Register Selector Logic



#### 2.4.2.2 Pointer Register Control Registers

In using the Pointer Registers several additional control registers are necessary: 15 Name Registers (NR's), 15 Shadow Name Registers (SNR's), 15 one-bit Name Flag Registers (NFR's), the Tag Register (TGR), and two counters (CCT and ACT). They are arranged as shown in Figure 2.4.2.2 along with the Name Bus (NB) and the Shadow Name Bus (SNB). All the registers (except for the NFR's), counters and buses are 4 bits in length

The SNR's, NFR's, CCT, ACT and the SNB are used in the process of name permutation whose explanation is deferred until Section 4.2.4.3.

As mentioned previously, the Name Registers are used to hold the current name for each PR. The PR's themselves each have a fixed, "wired-in" name (0 through 14) corresponding to the PR's position in the hardware. Each physical position however, can be assigned, via the name registers, any distinct name from 0 to 14 by loading this name into the appropriate NR.

The Tag Register is used to store the tag portion of an operand phrase. This gives the name which is to be searched for in the Name Registers.

The Name Bus is in reality the input to the compare circuit. The compare circuit compares whatever is on the Name Bus with the contents of the Name Registers and produces an output on one of 15 output lines corresponding to the Name Register which matches the input.

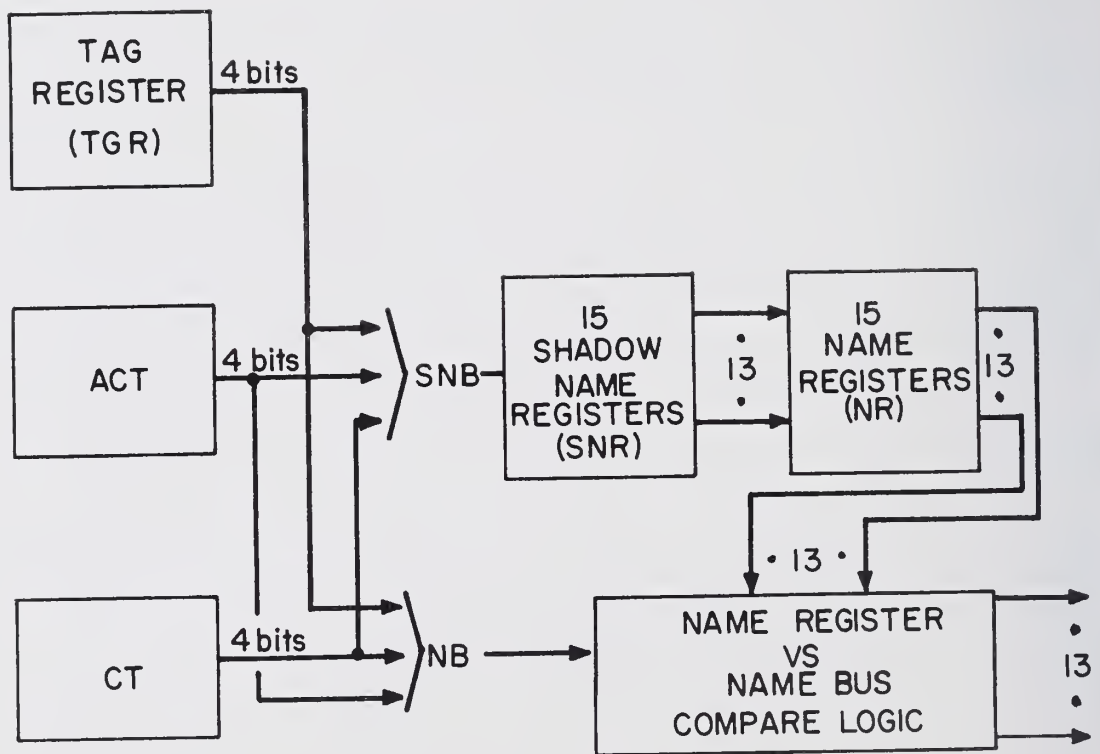


FIGURE 2.4.2.2 POINTER REGISTER CONTROL REGISTERS

### 2.4.2.3 Pointer Register Selection Process

When a Pointer Register must be accessed, its name is to be found in the tag portion of the operand phrase currently being decoded. Therefore the first step in the PR selection process is to gate the tag portion of the IR into the TGR where it is stored for possible later use.

The TGR is subsequently gated onto the Name Bus which is connected to the NR/NB Compare Logic. This logic has as inputs the Name Bus and the 15 NR's, and has as outputs, 15 select lines. The NB is compared with the NR's and the matching register activates an output on the select line corresponding to the physical PR with that name. For example, suppose the operand wants the PR currently named #3. This value is gated through to the NB and compared with all of the NR's. The NR with 3 as its contents then indicates which PR is #3 at the present time by activating the output line corresponding to that PR. If a match is obtained in none or more than one NR, a malfunction has occurred since the names in the NR's are supposed to be unique.

After the selected output line from the NR/NB Compare Logic is activated it is then used to activate the proper PR select line which is sent to the storage blocks and the segment name register selector and also to indicate which of the two storage groups (0 through 7 or 8 through 14) PR is in. This latter indication is of no importance when data is to be written into the storage group, since the writing process depends only on the word selected to be written into. On a read out, however, the output gates of the byte storage group which does not contain the selected word must be inhibited; otherwise, all zeros are output from this group. These zeros will in turn override any data output from the other group containing the selected cell.

The logic here senses for a select signal to one of the (physical) PR's 8 through 14 and enables the output select gate from that group. If one of PR's 8-14 is not selected, it is assumed that one of 0 through 7 is selected and that group's output gate is enabled. The group selection signals are PR7G/S and PR8G/S which select registers 0-7 and 8-14, respectively. In the rest, or inhibit, state, the PR8G/S rests '1', or in the select mode. This corresponds to the group containing the SBR. The select signal, when up, enables the output gates of the PR group selected. If no word internal to the 9 bit/byte storage group is also selected, the data output lines from the storage group will all rest '0'.

When the Spare Buffer Register (SBR) is to be used, the output selection signals from the NR/NB Compare Logic are inhibited so that no PR select lines are chosen. Therefore PR8G/S is on and PR7G/S is off. This causes the group containing the SBR to be chosen. The select signal for the SBR is given by the TP Control.



#### 2.4.2.4 Tag Setting and Sensing

Sense logic driven by the TGR is used to sense the contents of the TGR for PR#13. If PR#13 is selected (PR#13 corresponds to the Operand Stack Pointer), it may be necessary to clear out or initialize the hardware portion of the OS. This signal, N13S is routed back to the control logic; when N13S goes to '1', it indicates that PR#13 is being (or is to be) selected.

When the TGR, or either of the counters for that matter, is not selected to be gated onto the NB, the NB contains the number (name) 0000 (zero). This name corresponds to the instruction pointer register. Whenever it becomes necessary to access the instruction pointer, it is only necessary to inhibit the TGR to NB gating in order to select the PR named zero. Two other control signals NPR13/S and NPR14/S are used to force the names '1101' (PR#13) and '1110' (PR#14) onto the NB. Again in these cases, the TGR to NB gate is inhibited when the names are forced on to the NB. This allows PR#13, the OS pointer, or PR#14, the Available Space Pointer, to be selected without explicitly gating it into the TGR.



### 2.4.3 Signal Name Lists for Pointer Registers

#### 2.4.3.1 Control Signals

ACT/E/	-	ACT enable count (ACT = ACT + 1)
ANB/G	-	ACT → NB gate
ASB/G	-	ACT → SNB gate
BSR1/G/	-	interrupt recovery - load higher SNR's from DB
BSR2/G/	-	interrupt recovery - load lower SNR's from DB
CCT/E/	-	CCT enable count (CCT = CCT + 1)
CEQA	-	output - value of CCT = value of ACT
CNB/G	-	CCT → NB gate
CTSB/G	-	CCT → SNB gate
DBDR/G	-	DB → PR storage blocks
INB1/G/	-	interrupt save - gate lower SNR's onto BRØS
INB2/G/	-	interrupt save - gate higher NR's onto BRØS
IRTGO/G/	-	interrupt save - tag field in byte 0
IRTG1/G/	-	gate the IR → TGR, tag field in byte 1
IRTG2/G/	-	gate the IR → TGR, tag field in byte 2
NFCK/E/	-	name flag check enable
NFONE	-	output - name flag of SNR indicated by NB is = 1
NFR/G/	-	gate name flag register, i.e. set name flag register indicated by NCE.
NOS	-	Output - set to 1 if TGR = 0 (i.e. if instruction pointer is selected)
N13S	-	Output - set to 1 if TGR = 13 (i.e. if ØS pointer is selected)
NPRI3/S	-	Load NB with +13 (i.e. the operand stack pointer number)
NPRI4/S	-	Load NB with +14 (i.e. the available space pointer number)

PSNRi/S1 - PR Segment Name Register - Read Select Register i,  
 Signal 1  
 PSNRi/S2 - PR Segment Name Register - Read Select Register i,  
 Signal 2  
 PSNRR15/ - PR Segment Name Register #15 - Read  
 PSNRW15/ - PR Segment Name Register #15 - Write  
 PSNWi/S1 - PR Segment Name Register - Write Select Register i,  
 Signal 1  
 PSNWi/S2 - PR Segment Name Register - Write Select Register i,  
 Signal 2  
 RACT - Reset auxiliary 4 bit counter (ACT)  
 RCT - Reset 4 bit counter (CCT)  
 RNFR - Reset all name flag registers  
 RSNR/ - Reset all Shadow Name Registers to 0  
 SBR/S - Spare buffer register select  
 SBSR/G/ - Gate SNB → SNR(i) where i is given by NCE which  
 is on  
 SNNR/G/ - SNR → NR gate - (all registers whose name flags = 1)  
 SNRD/E/ - Enable PR Segment Name Register Read  
 SNS/G/ - Gate PR Select Signals, PRi/S/, into PR Segment  
 Name Selector Logic  
 SNWT/E/ - Enable PR Segment Name Register Write  
 TNB/G - Gate the TGR → NB  
 TSB/G - Gate the TGR → SNB  
 WRPRL/E - DBP → pointer link field selected by PRS's  
 WRPRV/E - DBP → SBR link field if SBR/S is on  
 WSBV/E - DBP → SBR value field if SBR/S is on

#### 2.4.3.2 Internal Signals Used by Pointer Registers

ACTi	-	Auxiliary 4 bit counter - bit i
BRØSi	-	Base register - Operand Stack bus bit i - to empty NR's
CCTi	-	4 bit counter - bit i
DBi	-	Permuter Output Bus - from DBP - ith bit
DBPi	-	Permuter Output Bus - directly from Permuter - bit i
NBi	-	Name Bus - bit i
NCEi	-	Name Compare Equal for ith register - i.e. NB = value in ith NR
NFRi	-	Name Flag Register - bit i
NjRi	-	jth name register - ith bit
PRBi	-	Pointer Register Bus output - bit i
Pri/S/	-	Pri has been selected
PR7G/S	-	PR selected belongs to PR's 0 through 7
PR8G/S	-	PR selected belongs to PR's 8 through 14 or SBR
PRSNBi	-	PR Segment Name Register Output Bus, bit i
PSNAi	-	PR Segment Name Register, Outbus A, bit i
PSNbi	-	PR Segment Name Register, Outbus B, bit i
SNBi	-	Shadow Name Bus - bit i
TGRi	-	Tag Register, holds tag from IR - bit i



```

// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  PREGIN: PROC(
    ACTF,      ANBG,      ASBG,      CCTF,      CEQA,      CNBG,
    CTSHG,     IMM,       INB1G,     INB2G,     N13S,     NFCKF,
    NFOINE,    NFRG,      NPR13S,    NPR14S,    RACT,     RCT,
    RNFR,      RSNR,      SBR,      SBSRG,     SMNRG,     TGIM,
    TNBG,      TSBG,      IRTGG,     PSNPRBG);
    DCL(
      ACTF,      ANBG,      ASBG,      CCTF,      CEQA,
      CNBG,      CTSHG,     IMM,       INB1G,     INB2G,     N13S,
      NFCKF,     NFOINE,    NFRG,      NPR13S,    NPR14S,    RACT,
      RCT,       RNFR,      RSNR,      SBR,      SBSRG,     SMNRG,
      TGIM,      TNBG,      TSBG) BIT(1), IRTGG(0:2) BIT(1);
    DCL PSNPRBG BIT(1);
    DCL ADD1 ENTRY((4)BIT(1)) EXTERNAL;
    DCL( (ACT,CCT,TGR)(4), NFR(0:14), (BROS,IR,PRB,SBR)(36),
      PRS(0:14), (NR,SNR)(0:14,4), PR(0:14,36)) BIT(1)
      EXTERNAL;
    DCL NCE(0:14) BIT(1),
      (NB,
      SNB
      )(4) BIT(1),
      (PR7GS,
      PR8GS
      ) BIT (1),
      (I,J)      FIXED BIN;

/* SET UP COUNTERS */
  IF RCT THEN CCT='0'B;
  IF RACT THEN ACT='0'B;
  IF ACTF THEN CALL ADD1(ACT);
  IF CCTF THEN CALL ADD1(CCT);
  DO I=1 TO 4;
    IF ACT(I)~=CCT(I) THEN GO TO SKIP1;
  END;
  CEQA='1'B;

/* LOAD NAME BUS AND/OR SHADOW NAME BUS */
  SKIP1:NB,SNB='0'B;
  IF IRTGG(0) THEN
    DO I=1 TO 4;
      TGR(I)=IR(I);
      TGIM=IR(8)&IMM;
    END;
  IF IRTGG(1) THEN
    DO I= 1 TO 4;
      TGR(I)=IR(I+9);
      TGIM=IR(17)&IMM;
    END;
  IF IRTGG(2) THEN
    DO I= 1 TO 4;
      TGR(I)=IR(I+18);
      TGIM=IR(26)&IMM;
    END;
  IF TNBG THEN NB=TGR;
  IF ANBG THEN NB=ACT;

```

```

IF CNBG THEN NB=CCT;
IF TSBG THEN SNB=TGR;
IF ASBG THEN SNB=ACT;
IF CTSBG THEN SNB=CCT;
IF NPR13S THEN DO;
    NB='1'B;
    NB(3)='0'B;
    END;
IF NPR14S THEN DO;
    DO I=1 TO 3;
        NB(I)='1'B;
    END;
    NB(4)='0'B;
END;
IF TGR(1)&TGR(2)&~TGR(3)&TGR(4) THEN N13S='1'B;
ELSE N13S='0'B;

/* COMPARE NAME BUS WITH EACH REGISTER */
DO I=0 TO 14;
    NCF(I)='0'B;
    DO J=1 TO 4;
        IF NR(I,J)~=NB(J) THEN GO TO NOTEQ;
    END;
    NCF(I)='1'B;
NOTEQ:END;

/* SET UP POINTER SELECT SIGNALS */
IF SBRS|PSNPRBG THEN PRS='0'B;
ELSE PRS=NCF;

/* CHECK FOR NAME FLAG RESETS */
IF RNER THEN NFR='0'B;

/* CHECK FOR GATING IN NAME FLAG REGISTER */
IF NFRG THEN DO I=0 TO 14;
    IF NCF(I) THEN NFR(I)='1'B;
END;

/* CHECK NAME FLAG */
NFRNE='1'B;
IF NECKE THEN DO I=0 TO 14;
    IF NCF(I)&NFR(I) THEN NFRNE='0'B;
END;

/* SET UP GROUP SELECT SIGNALS */
PR8GS='0'B;
DO I=8 TO 14;
    IF NCF(I) THEN PR8GS='1'B;
END;
IF SBRS THEN PR8GS='1'B;
PR7GS=~(PR8GS);

/* GATE POINTER TO POINTER BUS AS OUTPUT */
IF PR7GS THEN DO I=0 TO 7;
    IF PRS(I) THEN PRB(*)=PR(I,*);
END;
IF PR8GS THEN DO;
    DO I=8 TO 14;

```



```

        IF PRS(I) THEN PRB(*)=PR(I,*);
            END;
        IF SBRS THEN PRB=SBR;
        END;

/* INTERRUPT BUS */
    IF INB1G THEN DO I=0 TO 3;
        DO J=1 TO 4;
            BRDS(I*9+J)=NR(2*I,J);
            BRDS(I*9+J+4)=NR(2*I+1,J);
        END;
    END;
    IF INB2G THEN DO I=0 TO 2;
        DO J=1 TO 4;
            BRDS(I*9+J)=NR(2*I+8,J);
            BRDS(I*9+J+4)=NR(2*I+9,J);
        END;
        DO J=1 TO 4;
            BRDS(27+J)=NR(14,J);
            BRDS(31+J)='0'B;
        END;
    END;

/* CHECK FOR RESET OF SNR */
    IF RSNR THEN SNR='0'B;

/* CHECK FOR GATING(SNB TO SNR) */
    IF SBSRG THEN DO I=0 TO 14;
        IF NCE(I) THEN SNR(I,*)=SNB;
    END;

/* CHECK FOR SNR TO NR */
    IF SNMRG THEN DO I=0 TO 14;
        IF NER(I) THEN NR(I,*)=SNR(I,*);
    END;

END PREGIN;

```



## 2.5 Base Registers

The Base Registers (BR's) are used as a part of the memory accessing scheme to contain information about the segments of the current process. Up to  $2^{14}$  segments are allowed the programmer. But as there are only 7 usable base registers, only the data on the 7 most recently used segments is held within the TP. The remaining segment base information is retained in memory in the Segment Name Table of the current process. BR's 1 through 7 are used to address recent data. BR#0 is used to indicate the Segment Name Table.

To identify which segments have their bases in the TP, each base register (other than BR#0) has an associative register assigned to it which contains the name of the segment currently in that base register. When a PR is selected as an address of a cell to be accessed from memory, its segment name register is associatively compared with the associative registers for each base register. If a match is found, the matching base register will be used. If there is no match, the required base data will be accessed from the Segment Name Table.

In order to completely determine a file, the base register must contain:

- 1) a base address which indicates where the file storage begins. (In Partitioned Mode the base address is the address of the page map - see Section 4.1.3.2).
- 2) a bounds, which indicates how many pages have been assigned,
- 3) a code bit to indicate the Data Access Mode (Contiguous or Partitioned), and
- 4) two other code bits to determine the level of Accessing Privilege (read-write, read only, trap, or no access).

The base address uses a 16 bit page address which represents the upper 16 bits of a normal 24 bit address. The rightmost 8 bits of the corresponding absolute address are always taken to be zeros. The bounds is an 8 bit quantity allotting the user a specified number (up to 256) of pages of memory for this file. These two quantities are placed in a 3 byte BR with the bounds in the leftmost byte, the base address in the rightmost 2 bytes, as shown in Figure 2.5/1. The Data Access Mode bit is in the flag position of the leftmost byte and the Accessing Privilege bits are in the flag positions of the 2 rightmost bits. How these various quantities are used is explained in Section 4.1 on Memory Accessing.

Since the base information controls where and how a user may access the memory, users must be prevented from changing the contents of any BR. For this reason instructions which change the value of any portion of a BR are considered "privileged" and may only be used by the supervisor routine.

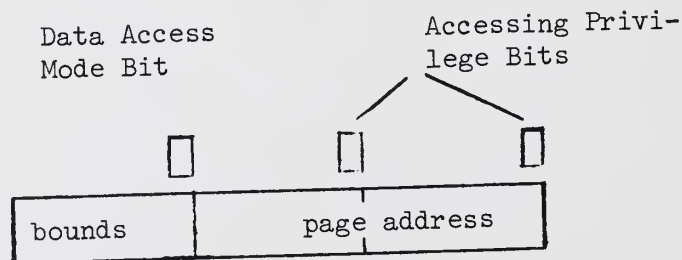


Figure 2.5/1 - Base Register Format

### 2.5.1 Base Register Functional Description

The Base Register Logic is shown in block diagram form in Figure 2.5.1. In a memory access a pointer register will be selected and one of the segment name registers will be gated out to the PRSN bus. This is then associatively compared with all of the Associative Registers. If there is a match, the corresponding BR select line will be activated and the proper Base Register will be read out onto the BROS bus. If there is no match, a special control sequence will be entered to access the Segment Name Table for the proper base information. This data is then stored in the base register which has been "inactive" for the longest amount of time. This choice of BR is made by the Queue Counter Logic (Section 2.5.1.2).

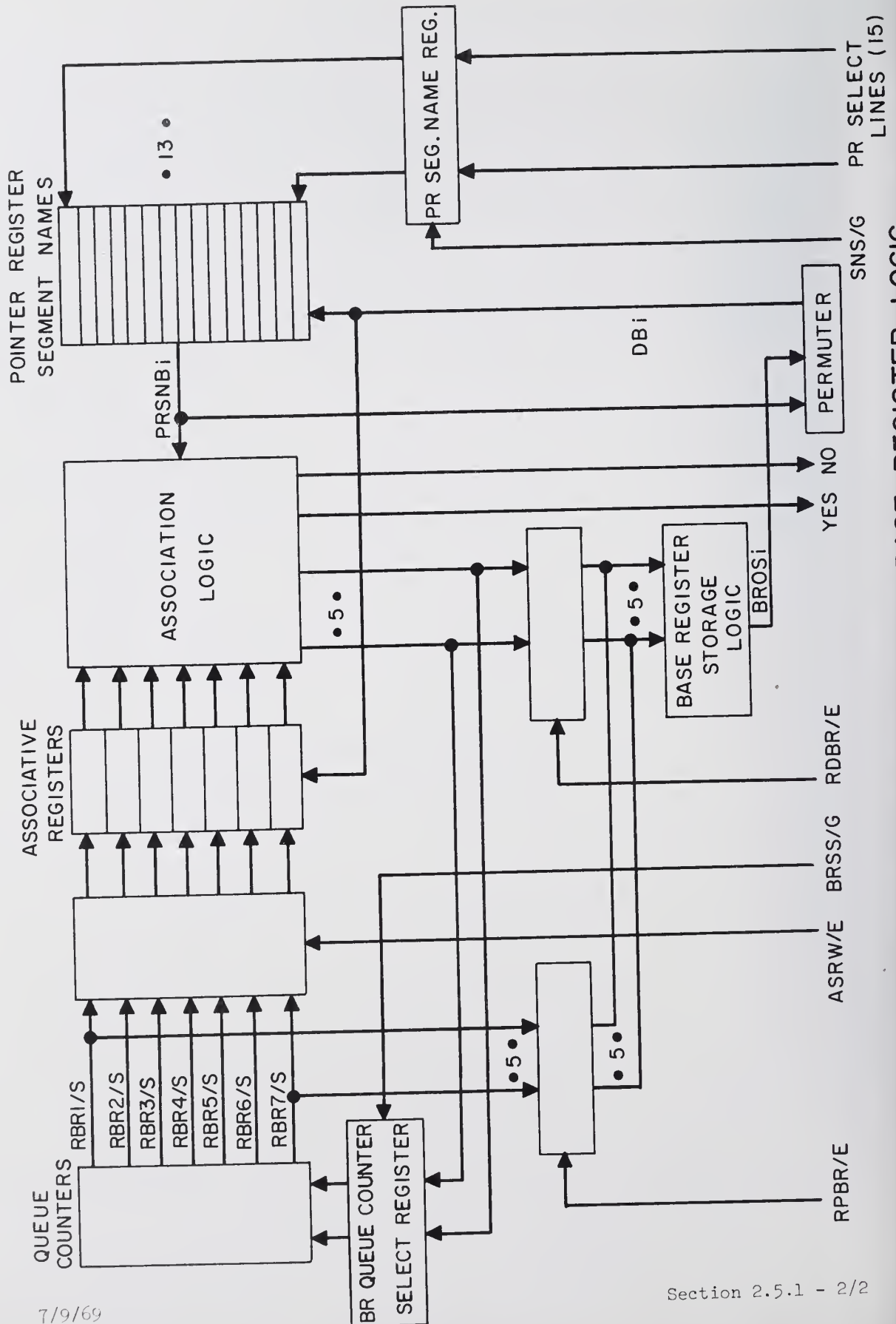


FIGURE 2.5.1 ASSOCIATIVE BASE REGISTER LOGIC

#### 2.5.1.1 Base Register Storage

The Base Register storage is made up of the same type of storage blocks as the Operand Stack and the Pointer Register, i.e. 9, 204-00 circuit boards driven by 2, 205-00 circuit boards. The BR storage consists of 4 blocks arranged as shown in Drawing Number 01-1 of the TP Logic Book. (For a detailed description of these blocks - see Section 2.2).

Because of the fact that the BR's are only 3 bytes long, they are staggered in their arrangement in the 4 subgroups. This arrangement is shown in Figure 2.5.1.1, where  $BR_{ij}$  indicates the position of the  $j$ th byte of the  $i$ th Base Register. Each Base/Bounds register is a 3 byte unit and the Byte Select inputs are common in 3 byte groups. Note that in this arrangement, there are 8 bytes of storage which remain unused. As will be explained later (in Section 2.6), these are used for the Instruction Buffer Register (IBR).

The inputs to the Base Register storage group consist of the Distribution Bus from the Permuter (DBP), the Base Register select lines, and read/write control lines. The Base Register select lines are connected to the word select inputs of the storage block Byte Select Drivers (BSD). The write operation takes two control lines,  $DBRS/G/$  to gate the DBP to the inputs of the flip-flops and  $WRBR/E$  which puts the selected flip-flops in the "write in" state so that they will accept the data gated in. The  $WRBR/E$  signal connects by means of an inverter to the read/write input of the BSD while the  $DBRS/G/$  connects directly to the "Gate Both In" line of the Input Gate Drivers.

Previous to gating the data into the flip-flops, the Permuter is used to shift the data so that when it is placed on the DBP, its boundaries will coincide with those of Figure 2.5.1.1.

Group 0	Group 1	Group 2	Group 3
BR01	BR02	BR03	BR11
BR12	BR13	BR21	BR22
BR23	BR31	BR32	BR33
BR41	BR42	BR43	BR51
BR52	BR53	BR61	BR62
BR63	BR71	BR72	BR73
RESERVED FOR INSTRUCTION BUFFER			
	REG.		

Figure 2.5.1.1 Arrangement of Base Registers

The output of the BR Storage Group is the Base Register-Operand Stack Bus (BROS). When the BRBSP/G signal, which is connected directly to the "Gate FalseOut" input of the Output Gate Drivers, is activated, the inverted contents of the selected BR is gated to the BROS. The data, at this point, is still in the position given in Figure 2.5.1.1. The Permuter can then be used to shift it so that it is delivered to the DB, left justified.



### 2.5.1.2 Associative Register Storage

The Associative Register Storage block is completely made up of IC chips. It consists of 7 registers each containing 16 bits, along with the appropriate input and output gating. An example of one register is shown in Figure 2.5.1.2/1.

Each Associative Register itself is made up of four SN7475 quad bistable latch chips. The input data comes from the DBB bus which is merely an extension of the DB bus into the IC logic part of the machine. The input gating signals are generated by 2 NOR circuits whose outputs go to 1 when the inputs,  $\overline{\text{ASRW/E}}$  and  $\overline{\text{RBRI/S}}$ , go to zero.  $\overline{\text{ASRW/E}}$  is the main signal for writing data into an Associative Register.  $\overline{\text{RBRI/S}}$  is a select signal generated by the Queue Counter logic (see Section 2.5.1.3). It selects the particular Associative Register to be written into.

Under normal program control there is never any need to read out the contents of an associative register since these registers are only used in comparisons with the PR Segment Name Bus. When a task is removed from a TP the Associative Register contents are simply abandoned by setting all of the OK bits to zero.

It is necessary, however, for the Engineering Console (see Section 4.7) to be able to display the contents of these registers. For this reason the Associative Registers have output gates which allow a selected register to be gated on to the PRSNB bus and from there to the PRB bus and the Permuter input gate.



### 2.5.1.3 Queue Counters

The Queue Counters are used to keep track of which base register has not been used for the longest length of time. The queue counter logic provides a series of output signals (REPLACE SELECT i) which indicate which associative register has not been in use for the longest period.

The overall idea is to use 3 bit counters to indicate a base register position in a service queue from 0 to 6. The register labeled 0 is that register most recently accessed, while the register labeled 6 was used least recently. Each time a memory access is performed (and the segment name is among those currently in the associative registers), the counter corresponding to the associative register is set to zero, and all the counters with counts less than the previous contents of the selected counter are incremented by one. This ensures that the order in the queue will always be in the order of most recent usage. Therefore the least recently used associative register will always have a counter containing a 6.

The detailed logic of the Queue Counters is shown in Figure 2.5.1.3./1. Operation is as follows.

Each counter has an overflow detector in the form of a flip-flop. After a base register-associative register combination has been selected, all of the 3 bit queue counters are incremented one count at a time until the counter corresponding to the selected register overflows. At this point an extra count is added to all counters whose overflow flip-flops are still off. Then the count pulses are continued until the total number of pulses, not counting the extra pulse, is 8. All counters with numbers  $\geq$  the count in the selected base register have now returned to their initial value, while those with initial counts the selected count have been incremented by 1. Also all overflow flip-flops are now on.

The overflow flip-flops are now reset by inserting a common count pulse into each overflow flip-flop. Finally the selected counter is reset to zero. An example of the counting is shown in Figure 2.5.1.3/2.

For the queue counters to function properly each counter must initially contain a distinct number from 0 to 6 inclusive. Therefore at turn-on a special sequence must be performed to ensure that this condition is met. Once the counters have been set up, however, they will continue to operate properly as long as no hardware error occurs.

The setup process is quite simple. It consists of using the cycle counter to select each queue counter in turn to be reset. Between each reset, all the queue counters are incremented once. Thus after 7 cycles there will be one counter with each number from 0 to 6. The logic is given in TP Logic Book, Drawing No. 23-3.

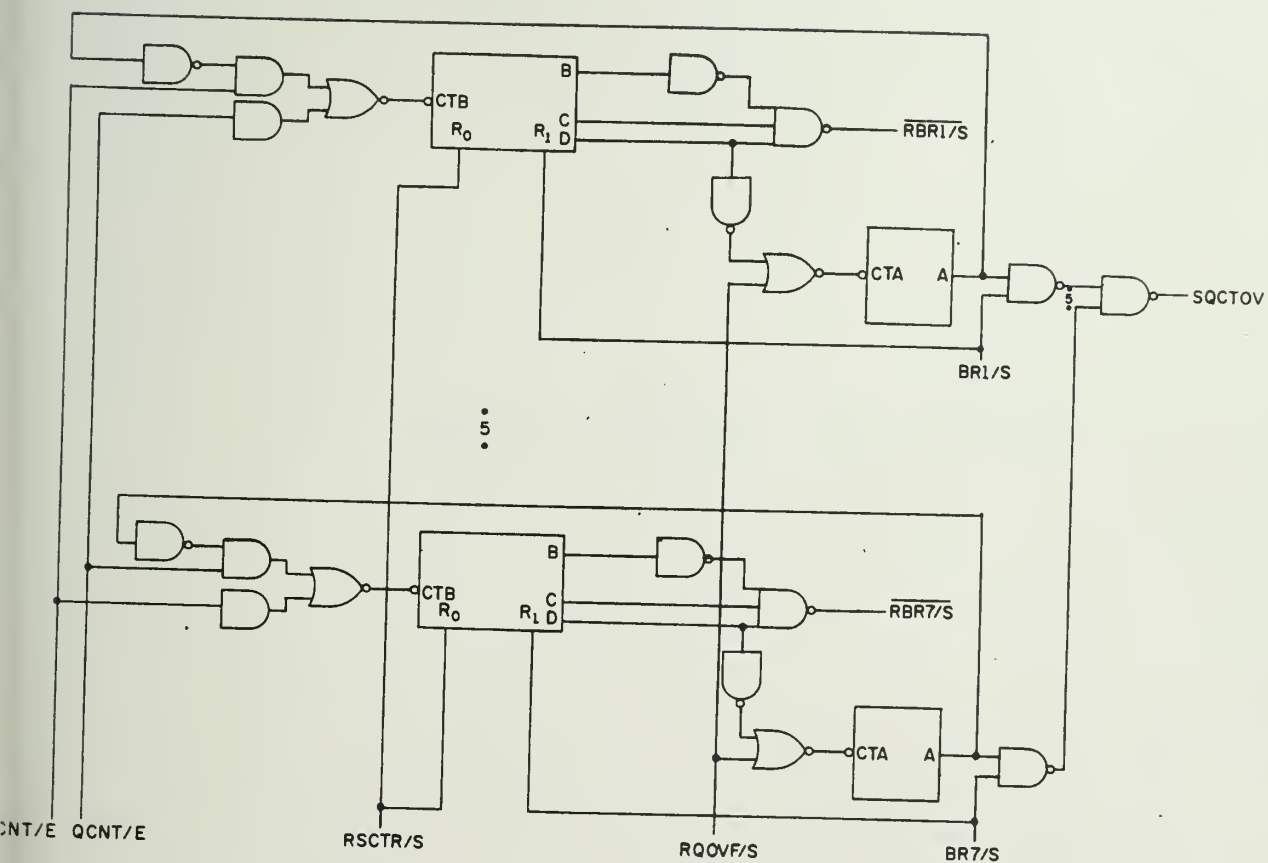


Figure 2.5.1.3/1 - Queue Counters

11/5/70

Section 2.5.1.3 - 3/4

Contents of Counters	Count Cycles									
	1	2	3	4	5	X	6	7	8	
0	1	2	3	4	5	6	7	0*	1	
1	2	3	4	5	6	7	0*	1	2	
2	3	4	5	6	7	0*	1	2	3	
Counter of Selected → 3	4	5	6	7	0*	0	1	2	③	← This is then reset to 0.
Associa- tive	4	5	6	7	0*	1	1	2	3	4
Register	5	6	7	0*	1	2	2	3	4	5
	6	7	0*	1	2	3	3	4	5	6

\* = overflow

X = extra count

Figure 2.5.1.3/2      Operation of the Queue  
Counters: An Example

#### 2.5.1.4 Association Logic and Associative Registers

The Association Logic compares the contents of the PRSNB bus with each of the Associative Registers "connected" to the BR's. It is conceptually similar to the comparison logic for matching the Name Bus with the Name Registers in the Pointer Register logic. The Associative Registers however are 16 bits long while the Name Registers are only 4 bits in length.

The association logic is shown in Figure 2.5.1.3/1. The OKn signal comes from a flip-flop attached to the particular association register which indicates that the data in the register is valid. Thus all that is needed to "empty" the registers is to set the seven OK flip-flops to 0.

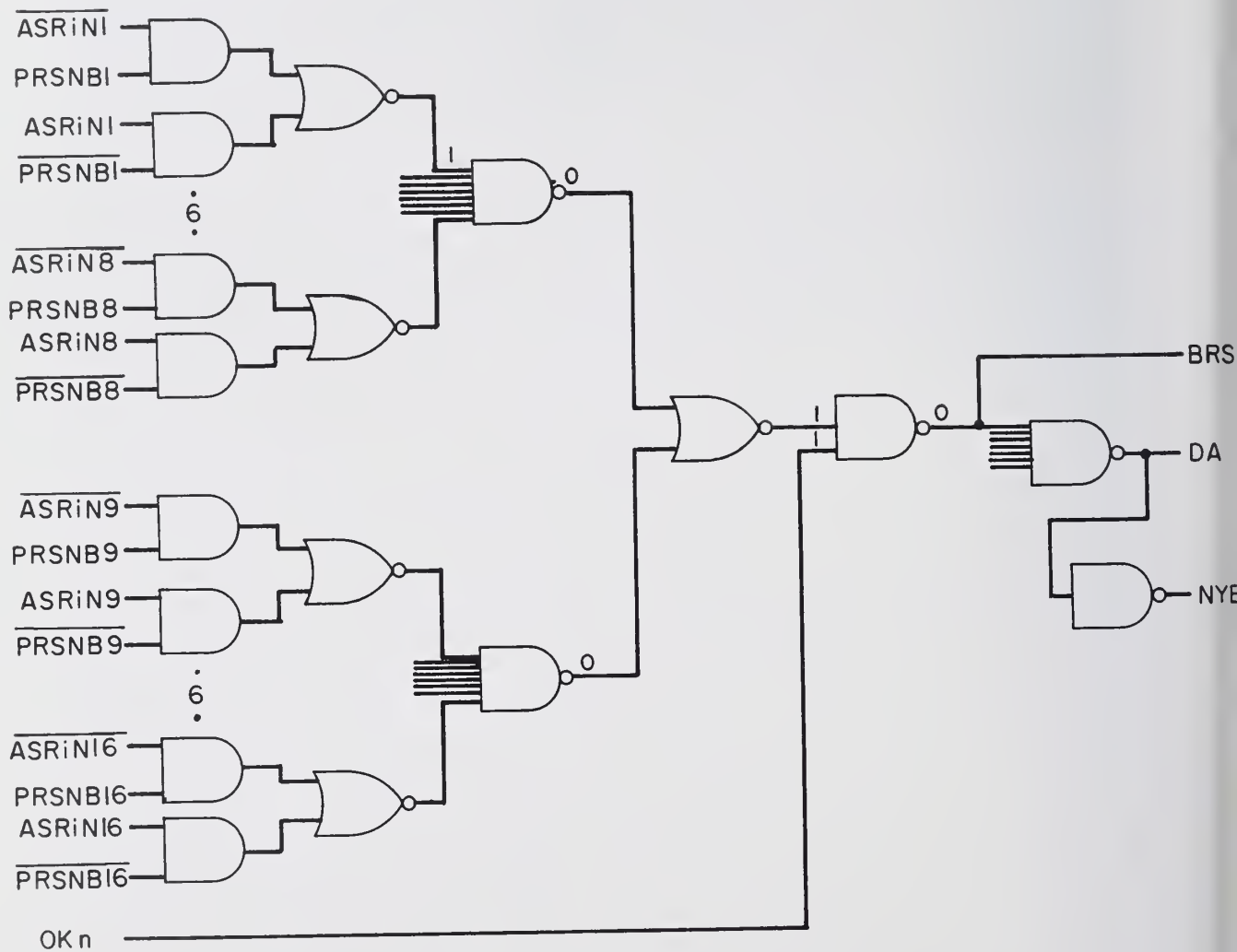


Figure 2.5.1.4/1 ASSOCIATION LOGIC FOR ASSOCIATION REGISTER i.



## 2.5.2 Signal Name Lists for the Base Registers

### 2.5.2.1 Control Signals

ASRiW/E1	Associative Register i, write enable 1
ASRiW/E2	" " " " " 2
ASRW/E	Associative Register write enable. Enables Associative Register selected by RBri/S signals.
BRG/S	Select BR portion of BR-IBR storage block.
Bri/S/	Select BRi in BR-IBR storage block.
BRRi/S/	BRi was selected by association logic
BRQi/S	Select BR Queue counter i
BRSBP/G/	Gate base register storage to BROSi according to BRi/S signals
BRSS/G	Gate BRNi/S signals to BR Queue counter select register.
CYC6	Output Signal - cycle counter equals 6.
CYCØV	Output Signal - cycle counter overflow
CYCT/E	Enable count of +1 on cycle counter.
DA	Output signal - a match between PRSNB and an associative register has been made.
NYET	Output signal - no match between PRSNB and an associative register
QCNT/E	Enable count of +1 on all Queue counters
QSETUP	Turned on during initial Queue counter set up procedure.
RBri/S/	Select BRi for reloading
RCYCT/S	Reset cycle counter
RPBR/E	Enable the replacement of the BR selected by the RBri/S signals
RQØVF/S	Reset overflow bit on Q counters. This signal only works if all overflow bits are on.
RSCTR/S	Reset selected Q counter
SNS/G	Gate PR Select Signals into select register
SQCTØV	Overflow by counter selected by BRi/S
XQCNT/E	Enable extra count of +1 for all Queue counters with their overflow bit off



## 2.5.2.2 Base Register Control - Internal Signals Used by the Base Registers

CMAi/	Compare result of first quarter of associative register i with PRSNB. Set to 0 if equal.
CMBi/	Compare result of second quarter of associative register i with PRSNB. Set to 0 if equal.
CMCi/	Compare result of third quarter of associative register i with PRSNB. Set to 0 if equal.
CMDi/	Compare result of fourth quarter of associative register i with PRSNB. Set to 0 if equal.
BR <del>O</del> Si	Output bus for Base Registers and Operand Stack.
DBBi	Permuter Output Bus used by control logic. Logically equivalent to DB, but produced by IC drivers, bit i
DBPi	Permuter Output bus direct from Permuter.
OKi	Associative Register i contains valid data.
PRSNBi	PR Segment Name Bus - bit i.



```

EXEC PL1
PL1.SYSPUNCH DD SYSOUT=B
PL1.SYSIN DD *
  BREGIN: PROC(BRGS,      BRSSG,      CYC6,      CYCOV,      CYCTE,
                DA,       NYET,       PSNRR15,   PSNRW15,   QCNTF,
                QSETUP,   RCYCTS,     RDBRE,     RPBRE,     RQOVFS,
                SNRDE,    SNSG,       SNWTE,     SQT0V,     WRBRE,
                XQCNTF,   BRS,       PSNPRBG);
  DCL(BRGS, BRSSG, CYC6, CYCOV, CYCTE, DA,
       NYET, PSNRR15, PSNRW15, QCNTF, QSETUP,
       RCYCTS, RDBRE, RPBRE, RQOVFS, SNRDE,
       SNSG, SNWTE, SQT0V, WRBRE, XQCNTF)
       BIT(1);
  DCL PSNPRBG BIT(1);
  DCL ADD1 ENTRY( (4) BIT(1)) EXTERNAL;
  DCL PRB(1:36) BIT(1) EXTERNAL;
  DCL( ASRG(7,18), BR(0:7,36), BROS(36), CYCTR(4),
       QCOUNT(7,4), PRSNB(1:36), PRSNR(0:15,18),
       (PRS, PSNRS1, PSNRS2, PSNWS1, PSNWS2, SNSRG)(0:15),
       (BRCS, BROS, BRRS, BRWS, CMA, CMB, CMC, CMD, OK, RBRs)
       (0:7)) BIT(1) EXTERNAL;
  DCL BRS(0:7) BIT(1);
  DCL (PSNA, PSNB)(1:17) BIT(1),
       COUNTER(4) BIT(1),
       (X, I, J) FIXED BIN;
  DCL BRSREG(0:7) BIT(1) EXTERNAL;

```

```
/* TURN ON ASSOCIATIVE LOGIC CONTROL SIGNALS */
```

```

RPRE=BRGS&WRBRE;
RDBRE=BRGS&~WRBRE;
BRSSG=RDBRE;

```

```
/* GATE PR SELECT SIGNALS INTO SELECT REGISTER SNSRG */
```

```

IF SNSG THEN SNSRG=PRS;
  IF SNRDE THEN DO I=0 TO 14;
    PSNRS1(I), PSNRS2(I)=SNSRG(I);
  END;
  IF SNWTE THEN DO I=0 TO 14;
    PSNWS1(I), PSNWS2(I)=SNSRG(I);
  END;
  PSNRS1(15), PSNRS2(15)=PSNRR15;
  PSNWS1(15), PSNWS2(15)=PSNRW15;

```

```
PSNA, PSNB='0'B;
```

```

DO J=1 TO 8;
  DO I=0 TO 7;
    PSNA(J)=PSNA(J)|(PSNRS1(I)&PRSNR(I,J));
  END;
  DO I=8 TO 15;
    PSNB(J)=PSNB(J)|(PSNRS1(I)&PRSNR(I,J));
  END;
END;

```

```

DO J=10 TO 17;
  DO I=0 TO 7;
    PSNA(J)=PSNA(J)|(PSNRS2(I)&PRSNR(I,J));
  END;
  DO I=8 TO 15;
    PSNB(J)=PSNB(J)|(PSNRS2(I)&PRSNR(I,J));
  END;
END;

```

/\* PR SEGMENT NAME REG OUTPUT BUS \*/

```

DO I=1 TO 17;
  PRSNB(I)=PSNA(I)|PSNB(I);
END;
IF PSNPRBG THEN PRB=PRSNB;

```

/\* COMPARE THE 4 QUARTERS OF EACH ASSOCIATIVE REGISTER WITH THE RESPECTIVE QUARTERS OF PRSNB. IF THE QUARTERS MATCH, SET THE RESPECTIVE CM SIGNALS TO 1 \*/

```

CMA,CMB,CMC,CMD='1'B;
DO I=1 TO 7;
  DO J=1 TO 4;
    IF PRSNB(J)~=ASRG(I,J) THEN CMA(I)='0'B;
  END;
  DO J=5 TO 8;
    IF PRSNB(J)~=ASRG(I,J) THEN CMB(I)='0'B;
  END;
  DO J=10 TO 13;
    IF PRSNB(J)~=ASRG(I,J) THEN CMC(I)='0'B;
  END;
  DO J=14 TO 17;
    IF PRSNB(J)~=ASRG(I,J) THEN CMD(I)='0'B;
  END;
END;

```

/\* BASE REGISTER SELECT SIGNAL DRIVERS \*/

```

BRRS='0'B;
BRWS='0'B;
IF RPBRE THEN DO I = 1 TO 7;
  BRWS(I)=BRRS(I);
END;
IF RDBRE THEN DO I=1 TO 7;
  BRRS(I)=OK(I)&CMA(I)&CMB(I)&CMC(I)&CMD(I);
END;
/*CALCULATE BRCS WHEN IMPLEMENTED */
BRS='0'B;
IF BRGS THEN DO I=1 TO 7;
  BRS(I)=BRRS(I)|BRWS(I);
END;
/* BRSREG IS THE REGISTER ON DRAWING 23-3 */
IF BRSSG THEN BRSREG=BRS;

```

```
/* CYCTR IS THE 4 BIT COUNTER ON DRAWING 23-3 */
```

```
IF CYCTR THEN CALL ADD1(CYCTR);  
IF RCYCTS THEN CYCTR='0'B;  
CYCONV='0'B;  
IF CYCTR(1) THEN CYCONV='1'B;
```

```
/* DECODE VALUE IN CYCTR */
```

```
X=0;  
IF CYCTR(2) THEN X=X+4;  
IF CYCTR(3) THEN X=X+2;  
IF CYCTR(4) THEN X=X+1;  
  
CYC6='0'B;  
IF X=6 THEN CYC6='1'B;  
  
IF QSETUP THEN BRQS(X+1)='1'B;  
ELSE DO I=1 TO 7;  
    BRQS(I)=BRSREG(I);  
END;
```

```
/* QCOUNT(I,J) IS THE ITH COUNTER JTH BIT OF THE INTERNAL  
COUNTERS ON DRAWINGS 23-1,2 */
```

```
IF RQOVFS THEN DO I=1 TO 7;  
    QCOUNT(I,1)=~QCOUNT(I,1);  
END;
```

```
/* INCREMENT COUNTERS AND CALCULATE RBRIS(I) SIGNALS */
```

```
DO I=1 TO 7;  
    IF (QCNTE|(XQCNTE&~QCOUNT(I,1))) THEN DO;  
        COUNTER=QCOUNT(I,*);  
        CALL ADD1(COUNTER);  
        QCOUNT(I,*)=COUNTER;  
    END;  
    RBRIS(I)=~QCOUNT(I,4)&QCOUNT(I,3)&QCOUNT(I,2);  
END;
```

```
/* CHECK IF COUNTER SELECTED BY BRS(I) HAS OVERFLOWED */
```

```
SQTOV='0'B;  
DO I=1 TO 7;  
    SQTOV=SQTOV|(BRQS(I) & QCOUNT(I,1));  
END;
```

```
DA='0'B;  
DO I=1 TO 7;  
    DA=DA|RBRIS(I);  
END;  
NYET=~DA;  
IF BRGS&~WRBRE THEN DO;
```

```
BROS='0'B;  
DO I=0 TO 7;  
    IF BRS(I) THEN BROS(*)=BROS(*)|BR(I,*);
```

```
END;
```

```
END;
```

```
END BREGIN;
```

```
/*
```



## 2.6 Instruction Buffer Register

The Instruction Buffer Register (IBR) is an eight byte (double word) register used to store the instructions as they are obtained from core. The Instruction Register (IR) is used to hold that part of the instruction which is currently being processed and is loaded from the IBR. In the case of Primitive Instructions the IR usually contains the complete instruction.

The IR is loaded from the IBR according to the count in the 3 bit Instruction Counter (ICT). This counter indicates the first byte of the next phrase to be worked on. When a new phrase is to be interpreted the byte designated by the ICT and the three succeeding bytes are read out on to the BROS bus from the fast registers.

In the next section a more detailed description of the operation of the IBR and ICT will be given.



## 2.6.1 Instruction Buffer Register-Functional Description

### 2.6.1.1 Instruction Buffer Register Storage

The Instruction Buffer Register is contained in the same 9 bit/byte storage blocks as the Base Registers (see Section 2.5.1.1). As can be seen in Drawing 01-1 of the TP Logic Book, each byte of the IBR has its own select line,  $\overline{\text{IBRO/S}}$ ,  $\overline{\text{IBR1/S}}$  etc. which is connected to the byte select input of the Byte Select Driver (BSD). The  $\overline{\text{WRIB/E}}$  line is used to indicate whether the selected byte should be read or written over.

In order to write,  $\overline{\text{WRIB/E}}$  must be "0",  $\overline{\text{DBBRS/G}}$  must be "0" in order to gate the data from the permuter to the flip-flop inputs, and the proper select lines must be "0".

The output of the IBR is connected to the BROS bus. When the  $\overline{\text{BRSBP/G}}$  signal, which is connected directly to the "Gate True Out" input of the Output Gate Drivers is activated, the four IBR bytes selected by the ICT are gated to the BROS. The permuter is then used to place them in sequential positions in the IR.



#### 2.6.1.2 The Instruction Counter and Selection Logic

The Instruction Counter (ICT) is a three bit double-rank counter which is capable of being counted up by ones or twos (refer to TP Logic Book Drawing 14-2). It has a special added bit to indicate overflow. The purpose of the ICT is to keep track of the initial byte position of the instruction currently being executed by the TP. For this reason it really represents the low order 3 bits of PR#0.

The ICT is used to drive the Instruction Buffer Register selection logic so that whenever the IR is loaded from the IBR the proper 4 bytes will be selected. The selection is done with a 234-04 diode matrix card, the same type as used in the Operand Stack selection logic (see Section 2.3.1.3). As shown in the TP Logic Book Drawing 14-1, the three output lines from the ICT can be gated to the input bus where they are decoded using eight 3-input NANDS from a 241-00 board. These eight outputs are then used as inputs into the 234-04. The diode matrix card activates the four desired select signals. Note that the 234-04 utilizes a wrap-around feature so that 4 bytes are always selected no matter what the value of ICT.

As can be seen in TP Logic Book Drawing 14-1, there is a provision for gating the last 3 bits of the DB into the ICT if the  $\overline{\text{DBCT/G}}$  line is on. This is used whenever PR#0 is loaded with a new value field. Whenever PR#0 is read out of storage, the 3 bits of the ICT are masked into the 3 low-order bit positions. This masking is performed using the ICTP/G signal and the OSR bus into the Permuter.

The overflow bit is used during instruction scanning. If it turns on, PR#0 must be incremented by 8.



## 2.6.2 Signal Name Lists for the Instruction Buffer Register

### 2.6.2.1 Control Signals

BRSBP/G/	-	BR Storage Block → Permuter Input
DBBRS/G/	-	Gate DB → BR-IBR Storage Block
DBCT/G/	-	Gate DB → ICT
IBR/S/	-	Selection IBR portion of BR-IRB storage
ICT1/E	-	Enable counter, $ICT = ICT + 1$
ICT2/E	-	Enable counter, $ICT = ICT + 2$
ICTP/G	-	Mask ICT into permuter output
WRIB/E/	-	Load instruction storage buffer read-write





#### 2.6.2.2 Internal Signals Used by the Instruction Buffer Register

DBi	-	Output bus from DBP - bit i
DBPi	-	Output bus directly from permuter - bit i
ICTi	-	Instruction Counter - bit i
IBRi/S/	-	Instruction Buffer - select byte i (4 chosen at a time)
IRi	-	Instruction Register - bit i
ØSRi	-	Bus from ØS registers to Permuter - bit i



## 2.6.4 Instruction Buffer Register - Logical Description

### 2.6.4.1 Instruction Counter Logic

The Instruction Counter (ICT), as shown in Drawing 14-2 of the TP Logic Book, is a three bit double rank counter which can count by one's or two's and which has a special overflow flip-flop to indicate when it has exceeded a count of 7. It can be set to a predetermined number by gating the true and complement values of the 3 low order data bits of the DB (DB33, DB34, and DB35) to the true and complement output lines of the upper rank of the counter. This causes the upper rank flip-flops to be forced to the value which appears on the 3 low order DB bit positions.

The ICT itself is a normal double rank counter in that the upper rank, which contains the current state of the counter, is gated into the lower rank during the time that the ICT is not counting (i.e. both ICT1/E and ICT2/E are '0'). The output of the lower rank, in turn is sent to the carry propagation logic which decides what the state of the ICT will be the next time that one of the count signals is activated. However since neither ICT1 or ICT2 is active at this time the actual value of the next state cannot be gated into the upper rank. When this does happen the gates which send the data from the upper rank to the lower rank will close. This ensures that the data inputs to the carry logic will not change. Then, depending on which signal was activated, the proper new state will be gated into the upper rank. When enough time has passed for the new count to be stable in the upper rank the count enable signal can be deactivated.

The carry generation logic is complicated by the fact that the ICT can be incremented by either one or two. In an incrementation by one, the low order ICT bit position, ICT3, will change state and the ICT2, ICT1 and ICTOV bit positions will change state if the previous state generated a carry.

In an incrementation by two, the second lowest order ICT bit position, ICT2, will change state and ICT1 and ICTOV will change state if the previous state generates a carry. In the count-by-two case ICT3 does not change state.

Thus what we have, in effect, is two sets of carry generation logic - one set to increment the counter by 1 and the other set to increment the counter by 2. Since the counter is only 3 bits long, full carry lookahead is used since it adds almost no extra logic in this case. All this means is that at any given bit position in the counter, that position will change state if every previous position in the counter is a '1' (for the case of incrementation by 1) or if every previous position with the possible exception of the lowest order position is a '1' (in the case of incrementation by 2).

Thus for the lowest order position, ICT3, all that is necessary is to change state when  $ICT1/E = 1$  and to do nothing when  $ICT2/E = 1$ . For the second lowest order position, ICT2, the upper rank flip-flop changes state every time  $ICT2/E = 1$  but if  $ICT1/E = 1$  it only changes state if ICT3 was also a 1. This process of changing state is accomplished by double gating the signals of the lower rank flip-flop into the upper rank flip-flop in such a manner as to reverse its setting. Only one of the gates will actually operate each time, one being used when the flip-flop is set to 1, the other when the flip-flop is set to 0.

The highest order position ICT1 is the most complicated of the three, but it is still fairly simple. If  $ICT2/E = 1$ , it will change state if ICT2 and ICT3 were both previously 1.

The ICT overflow bit, ICTOV, is set when ICT1 goes from a 1 state to a zero state since this only occurs when the ICT goes from '111' to '000'. The set signal is determined from the proper gating circuit for the ICT1 position.

It should be noted that in order to speed up the setting of the ICTOV flip-flop, the count signals  $ICT1/E$  and  $ICT2/E$  were used directly instead of using a doubly inverted gate signal as in the other ICT positions. This will not cause timing problems however since ICTOV does not feed back into a lower rank. It will enable a faster operation since the signals which indicate overflow will be valid at the time the count enable signals are activated.

## 2.7 The 32 Bit Adder

The 32 bit adder in the Taxicrinic Processor is used to perform binary addition within the TP. It is also used to generate the outputs for the boolean operations, EQV and XOR. The adder uses 2's complement number representation and employs two levels of carry lookahead to hasten carry propagation.\*

The inputs to the adder are obtained from the Distribution Register (DR) and the Permuter Distribution Bus (DB). These two inputs can be gated to the adder only in "true" form. To calculate the difference of two numbers, the subtrahend must be gated from the 9 bit/byte storage (Operand Stack, Pointer Registers, etc.) in 1's complement form (see Section 2.2). In this case a carry is injected at the low-order end of the adder to obtain the true 2's complement difference.

The adder itself is broken down into eight four-bit sum groups with full carry lookahead within the groups. The second level of lookahead occurs between groups, with lookahead between the leftmost groups 1, 2 and 3 and also between the rightmost groups 4 through 8. The final carry between these two, second level groups is a "ripple" carry.

The adder output is the  $\overline{\text{ARN}}$  bus. This is the input bus to the AR and must be latched into the Arithmetic Register (see Section 2.1.2.5 for a short description of a latch flip-flop). The  $\overline{\text{ARN}}$  is in complemented form, but converts to "true" form when it is latched into the AR.

---

\*For a more complete discussion of carry lookahead adders see Wiegel, Roger E., "Methods of Binary Addition", DCS Report No. 195, February 1966.



## 2.7.1 32 Bit Adder-Functional Description

### 2.7.1.1 Block Diagram Description

As can be seen from the block diagram in Figure 2.7.1.1/1 the 32 bit Adder consists of the following sections: Input Gates, Carry Generators, Propagation Generators, Group Carry Generators, and Output Gates. Each of these sections are represented by blocks in the diagram. Note that the 32 bit Adder is divided into eight 4-bit groups. The lines leaving the various sections are labeled with the number of signals they represent, either 1 or 4.

The purpose of the 8 Input Gates is to produce the carry transmit and carry generate signal for each bit position in a 4-bit group, i.e.

$$\overline{T_i} = X_i \cdot Y_i \vee \overline{X_i} \cdot \overline{Y_i} = X_i \oplus Y_i$$

$$G_i = \overline{X_i} \cdot Y_i$$

where  $X_i$  and  $Y_i$  are the  $i$ th bits of the DR and DB, respectively. If  $\overline{G_i} = 0$ , the  $i$ th bit position will generate a carry into the next (left) position,  $i-1$ . If  $\overline{T_i} = 0$ , the  $i$ th bit will not generate a carry, but it will transmit a carry if it receives one from the  $i + 1$  th position.

Each Input Gate generates 4  $\overline{T_i}$  and 4  $\overline{G_i}$  signals. The  $\overline{T_i}$  signals are sent to the Carry Generator, Propagate Generator and the Output Gates. The  $\overline{G_i}$  signals are only sent to the Carry Generator.

The Carry Generator and Propagate Generator comprise the first level of lookahead. The Carry Generator is used to generate the carry signals for each position in each four bit group. The Boolean expressions for the  $j$ th group are as follows:

$$C4G_j = CIN_j$$

$$C3G_j = G4 \vee T4 \cdot CIN_j$$

$$C2G_j = G3 \vee T3 \cdot G4 \vee T3 \cdot T4 \cdot CIN_j$$

$$C1G_j = G2 \vee T2 \cdot G3 \vee T2 \cdot T3 \cdot G4 \vee T2 \cdot T3 \cdot T4 \cdot CIN_j$$

$$COUT_j = G1 \vee T1 \cdot G2 \vee T1 \cdot T2 \cdot G3 \vee T1 \cdot T2 \cdot T3 \cdot G4$$

where  $T1$  through  $T4$  and  $G1$  through  $G4$  represent the  $T_i$  and  $G_i$  signals for the first through fourth bit positions in the  $j$ th group, respectively,  $CIN_j$  represents the carry into the  $j$ th group from the next lower order group, and  $COUT_j$  represents the carry out of the  $j$ th group into the next higher order group.



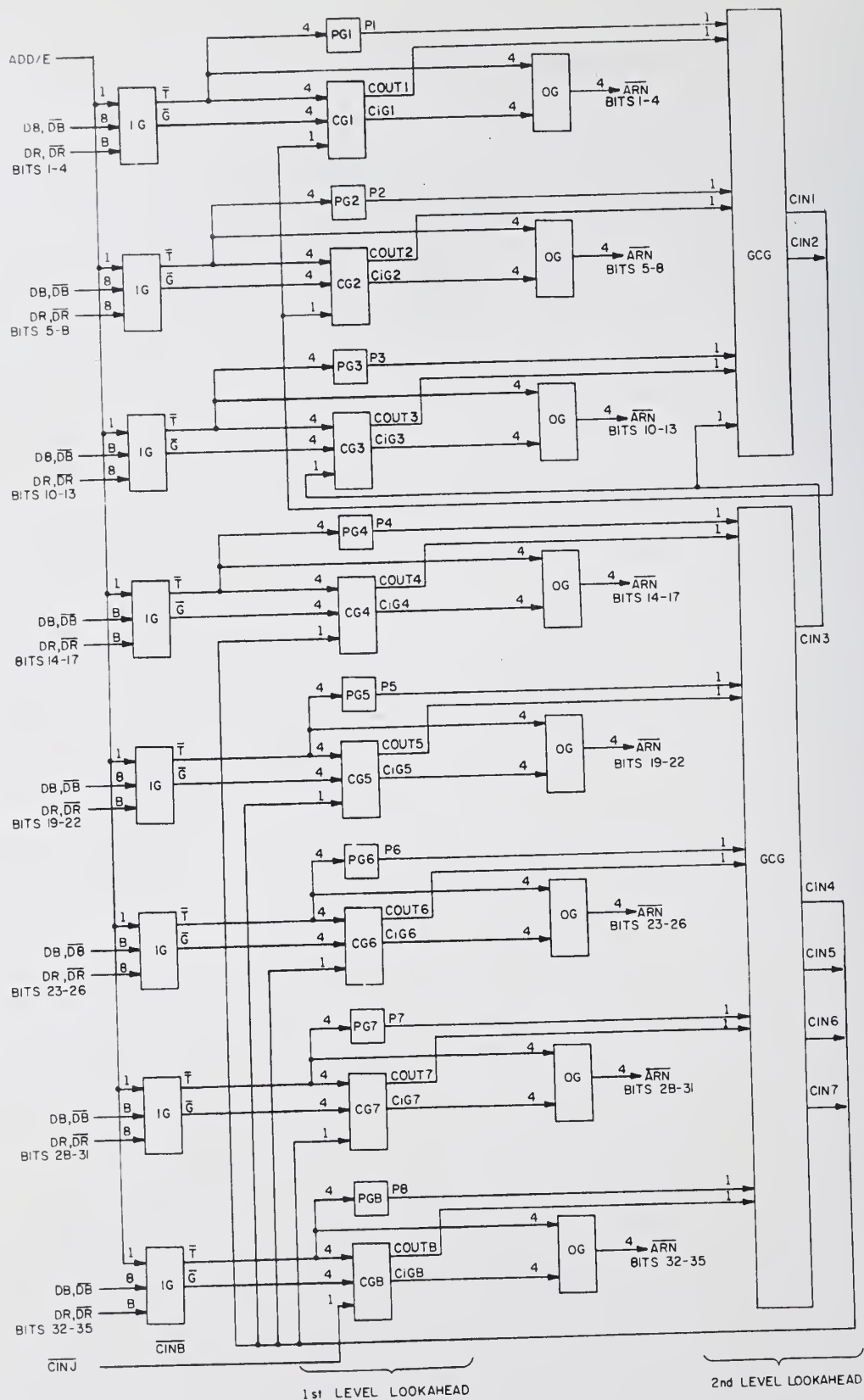


Figure 2.7.1.1/1 - TP - 32 Bit Adder



Note that the carries are produced by a carry generate signal in some position to the right of the position of interest with an unbroken series of carry transmit signals between the two positions. Note also that except for the last signal, which is actually a carry generate signal for the four bit group, all of the expressions depend on the input carry which will come from the previous group and which will not be known until later. However, at the first level of lookahead, we are really only interested in finding out if this group will generate a carry by itself, so only the last equation is important at this stage.

These carry signals are produced by using a diode matrix board. In order to facilitate the hardware realization and to speed up the adder, De Morgan's theorem was used to rewrite them in the following form:

$$\overline{C4j} = \overline{CINj}$$

$$\overline{C3j} = \overline{G4 \cdot T4} \vee \overline{G4 \cdot CINj}$$

$$\overline{C2j} = \overline{G3 \cdot T3} \vee \overline{G3 \cdot G4 \cdot T4} \vee \overline{G3 \cdot G4 \cdot CINj}$$

$$\overline{C1j} = \overline{G2 \cdot T2} \vee \overline{G2 \cdot G3 \cdot T3} \vee \overline{G2 \cdot G3 \cdot G4 \cdot T4} \vee \overline{G2 \cdot G3 \cdot G4 \cdot CINj}$$

$$\overline{COUTj} = \overline{G1 \cdot T1} \vee \overline{G1 \cdot G2 \cdot T2} \vee \overline{G1 \cdot G2 \cdot G3 \cdot T3} \vee \overline{G1 \cdot G2 \cdot G3 \cdot G4}$$

The logical drawing for the diode matrix is shown in Figure 2.7.1.1/2 while the diode layout is shown in Figure 2.7.1.1/3.

The Propagate Generator produces a propagate signal which indicates if the particular four-bit group will conduct an input carry all the way through the group. It is calculated from the  $\overline{Ti}$  signals of the four-bit group, i.e.

$$Pj = T1 \cdot T2 \cdot T3 \cdot T4 = \overline{\overline{T1} \vee \overline{T2} \vee \overline{T3} \vee \overline{T4}}$$

Once the  $Pj$  and  $COUTj$  signals have been produced, the first level of lookahead is completed. Note that the production of these signals only depends on the  $Ti$  and  $Gi$  signals which were produced by the Input Gates.

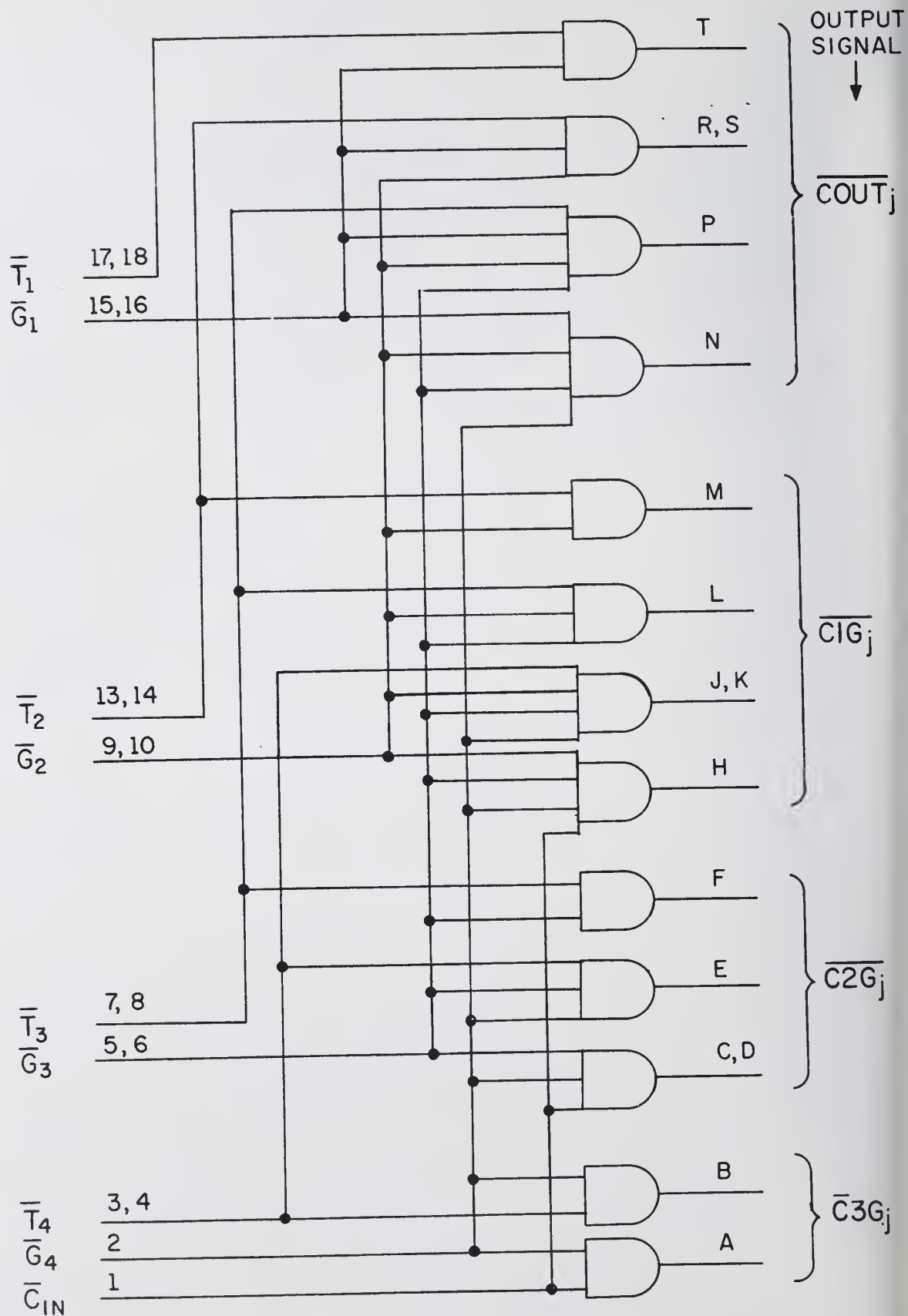


Figure 2.7.1.1/2 - Lookahead Carry Generator for Group  $j$

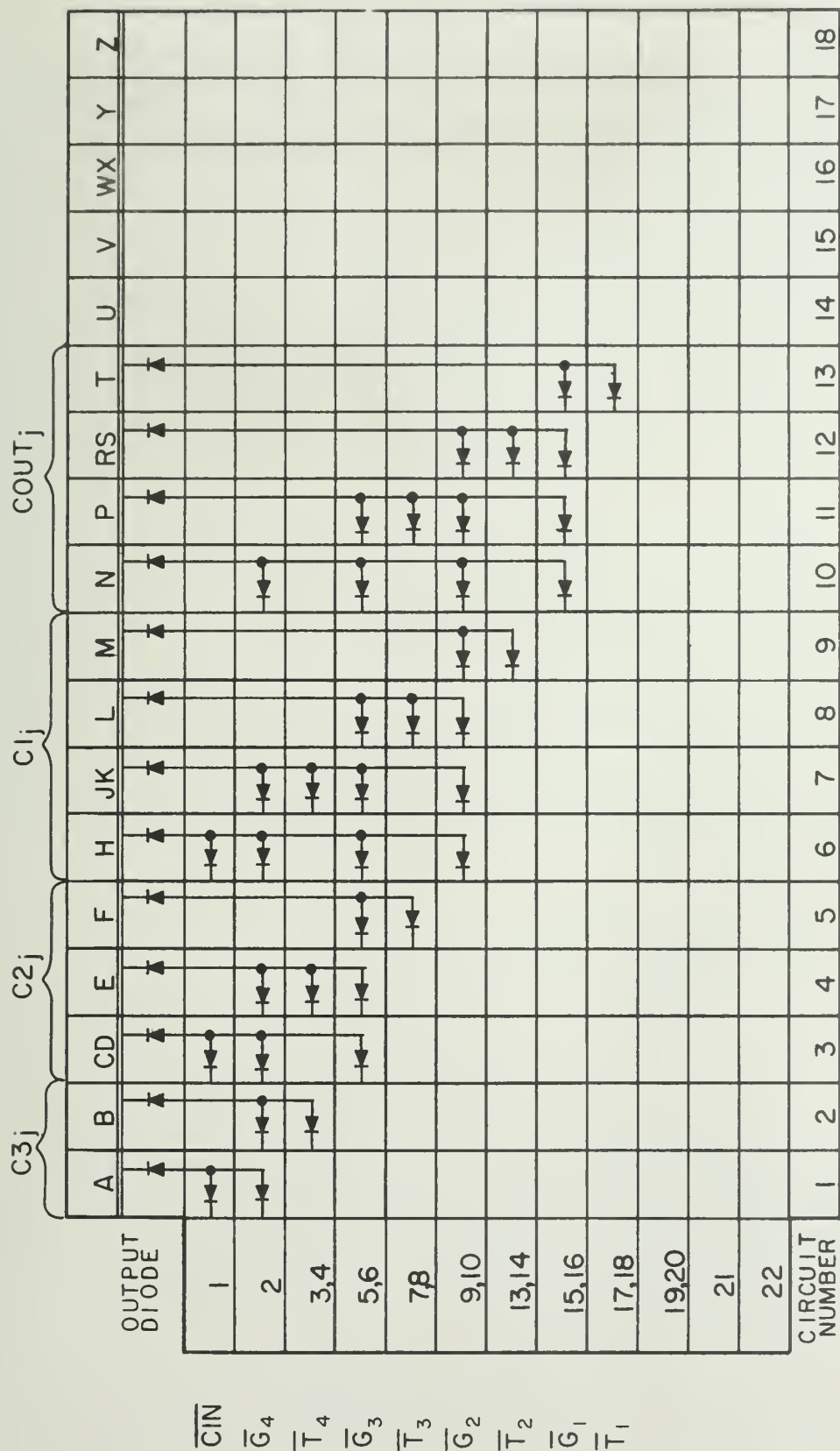


Figure 2.7.1.1/3 - 1st Level Lookahead Carry Generator - 236-14

The second stage of lookahead involves the Group Carry Generators. We want to calculate the CIN<sub>j</sub> signals for each group so that we can complete the calculation of the carry signals in the equations given above. Taking the low order Group Carry Generator first, we have the following equations:

$$\text{CIN}_8 = \text{CIN}_J$$

$$\text{CIN}_7 = \text{C}_8 \vee \text{P}_8 \cdot \text{CIN}_J$$

$$\text{CIN}_6 = \text{C}_7 \vee \text{P}_7 \cdot \text{C}_8 \vee \text{P}_7 \cdot \text{P}_8 \cdot \text{CIN}_J$$

$$\text{CIN}_5 = \text{C}_6 \vee \text{P}_6 \cdot \text{C}_7 \vee \text{P}_6 \cdot \text{P}_7 \cdot \text{C}_8 \vee \text{P}_6 \cdot \text{P}_7 \cdot \text{P}_8 \cdot \text{CIN}_J$$

$$\text{CIN}_4 = \text{C}_5 \vee \text{P}_5 \cdot \text{C}_6 \vee \text{P}_5 \cdot \text{P}_6 \cdot \text{C}_7 \vee \text{P}_5 \cdot \text{P}_6 \cdot \text{P}_7 \cdot \text{C}_8 \vee \text{P}_5 \cdot \text{P}_6 \cdot \text{P}_7 \cdot \text{P}_8 \cdot \text{CIN}_J$$

$$\begin{aligned} \text{CIN}_3 = & \text{C}_4 \vee \text{P}_4 \cdot \text{C}_5 \vee \text{P}_4 \cdot \text{P}_5 \cdot \text{C}_6 \vee \text{P}_4 \cdot \text{P}_5 \cdot \text{P}_6 \cdot \text{C}_7 \vee \text{P}_4 \cdot \text{P}_5 \cdot \text{P}_6 \cdot \text{P}_7 \cdot \text{C}_8 \\ & \vee \text{P}_4 \cdot \text{P}_5 \cdot \text{P}_6 \cdot \text{P}_7 \cdot \text{P}_8 \cdot \text{CIN}_J \end{aligned}$$

where CIN<sub>J</sub> is the low-order injected carry to the adder, and the C<sub>0j</sub>'s are the COUT<sub>j</sub> signals from the Carry Generators. Note that the form of the equations is the same as in the equations for the Carry Generator but in this case the C<sub>0j</sub> signals represent a carry generated by the group and the P<sub>j</sub> signals represent a carry transmitted by the group.

The 0-3 Group Carry Generator has to wait for CIN<sub>3</sub> to be formed before it can "start" since in its equations CIN<sub>3</sub> takes the place of CIN<sub>J</sub>, i.e.:

$$\text{CIN}_2 = \text{C}_3 \vee \text{P}_3 \cdot \text{CIN}_3$$

$$\text{CIN}_1 = \text{C}_2 \vee \text{P}_2 \cdot \text{C}_3 \vee \text{P}_2 \cdot \text{P}_3 \cdot \text{CIN}_3$$

This wait is what is referred to as the "ripple" carry in the adder.

Finally, with all the input carries to the groups determined, the carry generators can now generate the proper individual carries for each bit simultaneously and the Output Gates can use these carries and the  $\overline{T_i}$  signals previously calculated, to determine the adder output. These output signals are calculated for each bit position *i*, as follows:

$$\text{ARN}_i = T_i \cdot C_i \vee \overline{T_i} \cdot C_i$$

Overflow-Underflow signals are formed using sign bit and output information. This is explained in greater detail in Section 2.7.1.3.

#### 2.7.1.2 Adder Timing

Figure 2.7.1.2 gives a timing chart for the adder. Time is given in delay units in the left-hand column: 1 delay unit per NAND and 1/2 delay unit per AND-OR diode matrix. The event column is subdivided into sum groups 1 and 2 and groups 3-8, (where group 8 contains bits 32-35). The Sum in groups 1 and 2 takes 2 delays more to form than the sum in 3-8 due to the ripple-carry between second level lookahead units.

# EVENT

Sum Group:  
3, 4, 5, 6, 7, 8

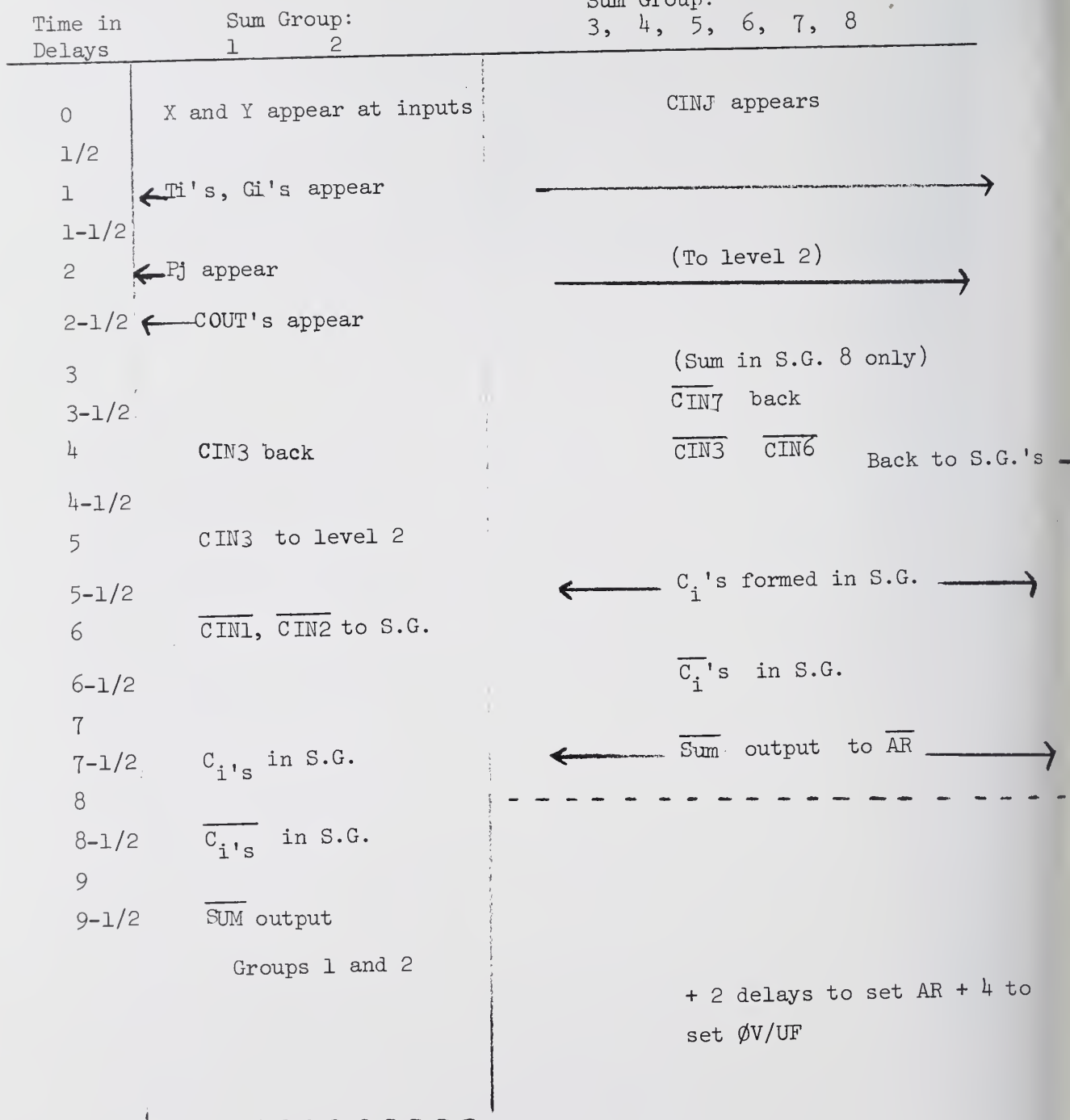


Figure 2.7.1.2 Lookahead Adder Timing

### 2.7.1.3 Adder Overflow

In considering the overflow conditions of the adder there are eight possible sign configurations which can occur:

B: Augend:	+	+	*	*	-	-
D: Addend:	+	+	-	-	-	-
A: Sum:	+	-	+	-	+	-
	00	01	10	01	10	11

\*2 cases each

The second and fifth cases are the error conditions. Overflow and underflow may be illustrated by two examples, respectively:

+5	0101
<u>+4</u>	<u>0100</u>
-7	1001
-5	1011
<u>-4</u>	<u>1100</u>
+7	0111

An overflow (or underflow) signal is given by:

$$\emptyset V = D \cdot B \cdot \bar{A} \vee \bar{D} \cdot \bar{B} \cdot A$$

as determined by the sign bits of the Augend (B), Addend (D) and Sum (A)





## 2.7.2 Signal Name Lists for the 32 Bit Adder Logic

### 2.7.2.1 32 Bit Adder Control Signals

ADD/E    -    Enable 32 bit adder  
BAR/G    -    Gates DB to AR through ARN



#### 2.7.2.2 32 Bit Adder Internal Signals

- ARNi - AR input bus - bit i. Output of 32 bit adder is loaded onto this bus.
- CINi/ - Carry into ith adder group, i = 3, ..., 7
- CINJ - Inject carry into low order bit of 32 bit adder
- COUTi - Carry generated within ith adder group, i = 2...,8  
In adder this represents ability of a given bit position to transmit an incoming carry.
- PRi - If on, indicates a carry will propagate through ith adder group, i = 3, ..., 8
- Ti/ - Equivalence function (NOT - XOR) between DBi and DRI.  
Used by both adder and Boolean logic.



```

// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
ADD32:  PROC(ADDE,CINJ, EQVE, XORE, ADDOUT);
        DCL(ADDE, CINJ, EQVE,XORF) BIT(1);
        DCL ADDOUT BIT(1);
        DCL(ARN,DB,DR,T)(36) BIT(1) EXTERNAL;
        DCL G(36) BIT(1) EXTERNAL,
          G1 BIT(1) DEFINED G POSITION(1 ),
          G2 BIT(1) DEFINED G POSITION(2 ),
          G3 BIT(1) DEFINED G POSITION(3 ),
          G4 BIT(1) DEFINED G POSITION(4 ),
          G5 BIT(1) DEFINED G POSITION(5 ),
          G6 BIT(1) DEFINED G POSITION(6 ),
          G7 BIT(1) DEFINED G POSITION(7 ),
          G8 BIT(1) DEFINED G POSITION(8 ),
          G9 BIT(1) DEFINED G POSITION(9 ),
          G10 BIT(1) DEFINED G POSITION(10),
          G11 BIT(1) DEFINED G POSITION(11),
          G12 BIT(1) DEFINED G POSITION(12),
          G13 BIT(1) DEFINED G POSITION(13),
          G14 BIT(1) DEFINED G POSITION(14),
          G15 BIT(1) DEFINED G POSITION(15),
          G16 BIT(1) DEFINED G POSITION(16),
          G17 BIT(1) DEFINED G POSITION(17),
          G19 BIT(1) DEFINED G POSITION(19),
          G20 BIT(1) DEFINED G POSITION(20),
          G21 BIT(1) DEFINED G POSITION(21),
          G22 BIT(1) DEFINED G POSITION(22),
          G23 BIT(1) DEFINED G POSITION(23),
          G24 BIT(1) DEFINED G POSITION(24),
          G25 BIT(1) DEFINED G POSITION(25),
          G26 BIT(1) DEFINED G POSITION(26),
          G28 BIT(1) DEFINED G POSITION(28),
          G29 BIT(1) DEFINED G POSITION(29),
          G30 BIT(1) DEFINED G POSITION(30),
          G31 BIT(1) DEFINED G POSITION(31),
          G32 BIT(1) DEFINED G POSITION(32),
          G33 BIT(1) DEFINED G POSITION(33),
          G34 BIT(1) DEFINED G POSITION(34),
          G35 BIT(1) DEFINED G POSITION(35),
          T1 BIT(1) DEFINED T POSITION(1 ),
          T2 BIT(1) DEFINED T POSITION(2 ),
          T3 BIT(1) DEFINED T POSITION(3 ),
          T4 BIT(1) DEFINED T POSITION(4 ),
          T5 BIT(1) DEFINED T POSITION(5 ),
          T6 BIT(1) DEFINED T POSITION(6 ),
          T7 BIT(1) DEFINED T POSITION(7 ),
          T8 BIT(1) DEFINED T POSITION(8 ),
          T10 BIT(1) DEFINED T POSITION(10),
          T11 BIT(1) DEFINED T POSITION(11),
          T12 BIT(1) DEFINED T POSITION(12),
          T13 BIT(1) DEFINED T POSITION(13),
          T14 BIT(1) DEFINED T POSITION(14),
          T15 BIT(1) DEFINED T POSITION(15),
          T16 BIT(1) DEFINED T POSITION(16),
          T17 BIT(1) DEFINED T POSITION(17),

```

```

T19 BIT(1) DEFINED T POSITION(19),
T20 BIT(1) DEFINED T POSITION(20),
T21 BIT(1) DEFINED T POSITION(21),
T22 BIT(1) DEFINED T POSITION(22),
T23 BIT(1) DEFINED T POSITION(23),
T24 BIT(1) DEFINED T POSITION(24),
T25 BIT(1) DEFINED T POSITION(25),
T26 BIT(1) DEFINED T POSITION(26),
T28 BIT(1) DEFINED T POSITION(28),
T29 BIT(1) DEFINED T POSITION(29),
T30 BIT(1) DEFINED T POSITION(30),
T31 BIT(1) DEFINED T POSITION(31),
T32 BIT(1) DEFINED T POSITION(32),
T33 BIT(1) DEFINED T POSITION(33),
T34 BIT(1) DEFINED T POSITION(34),
T35 BIT(1) DEFINED T POSITION(35);
OCL ( /*INTERNAL VARIABLES,*/
/* CARRY PROPAGATE SIGNALS*/
P1,P2,P3,P4,P5,P6,P7,P8,
/* CARRY INTO ITH ADDER GROUP */
CIN1,CIN2,CIN3,CIN4,CIN5,CIN6,CIN7,CIN8,
/* CARRY GENERATED WITHIN ITH ADDR GROUP*/
COUT1,COUT2,COUT3,COUT4,COUT5,COUT6,COUT7,COUT8,
/* CARRY INTO FIRST POSITION OF ITH ADDER GROUP*/
C1G1,C1G2,C1G3,C1G4,C1G5,C1G6,C1G7,C1G8,
/*CARRY INTO SECOND POSITION OF ITH ADDER GROUP*/
C2G1,C2G2,C2G3,C2G4,C2G5,C2G6,C2G7,C2G8,
/*CARRY INTO THIRD POSITION OF ITH ADDER GROUP*/
C3G1,C3G2,C3G3,C3G4,C3G5,C3G6,C3G7,C3G8,
/*CARRY INTO FOURTH POSITION OF ITH ADDER GROUP*/
C4G1,C4G2,C4G3,C4G4,C4G5,C4G6,C4G7,C4G8
) BIT(1);
PUT LIST('ADD32 ENTERED');
/* CALCULATE THE CARRY TRANSMIT SIGNALS (T) AND CARRY GENERATE
SIGNALS (G) */
IF EQVE|ADDF|XORE THEN DO J=0 TO 3;
DO I=J*9+1 TO J*9+8;
T(I)=DB(I)&~DR(I)|~DB(I)&DR(I);
G(I)=DB(I)&DR(I);
END;
END;
COUT1=G1|T1&G2|T1&T2&G3|T1&T2&T3&G4;
COUT2=G5|T5&G6|T5&T6&G7|T5&T6&T7&G8;
COUT3=G10|T10&G11|T10&T11&G12|T10&T11&T12&G13;
COUT4=G14|T14&G15|T14&T15&G16|T14&T15&T16&G17;
COUT5=G19|T19&G20|T19&T20&G21|T19&T20&T21&G22;
COUT6=G23|T23&G24|T23&T24&G25|T23&T24&T25&G26;
COUT7=G28|T28&G29|T28&T29&G30|T28&T29&T30&G31;
COUT8=G32|T32&G33|T32&T33&G34|T32&T33&T34&G35;
/* THE PROPAGATE GENERATOR: THE P SIGNALS INDICATE WHETHER THE
4TH BIT GROUP WILL CONDUCT AN INPUT CARRY ALL THE WAY
THROUGH THE GROUP */
P1=T1&T2&T3&T4;
P2=T5&T6&T7&T8;
P3=T10&T11&T12&T13;
P4=T14&T15&T16&T17;
P5=T19&T20&T21&T22;

```

```

P6=T23&T24&T25&T26;
P7=T28&T29&T30&T31;
P8=T32&T33&T34&T35;
CIN3= COUT4
      |COUT5&P4
      |COUT6&P4&P5
      |COUT7&P4&P5&P6
      |COUT8&P4&P5&P6
      |CINJ &P4&P5&P6&P7&P8;
CIN4= COUT5
      |COUT6&P5
      |COUT7&P5&P6
      |COUT8&P5&P6&P7
      |CINJ &P5&P6&P7&P8;
CIN5= COUT6
      |COUT7&P6
      |COUT8&P6&P7
      |CINJ &P6&P7&P8;
CIN6= COUT7
      |COUT8&P7
      |CINJ &P7&P8;
CIN7= COUT8
      |CINJ&P8;
CIN8=CINJ;
/*CALCULATE RIPPLE CARRY */
CIN2= COUT3|CIN3&P3;
CIN1= COUT2|COUT3&P2|CIN3&P2&P3;
C1G1=G2|T2&G3|T2&T3&G4|T2&T3&T4&CIN1;
C2G1=G3|T3&G4|T3&T4&CIN1;
C3G1=G4|T4&CIN1;
C4G1=CIN1;
C1G2=G6|T6&G7|T6&T7&G8|T6&T7&T8&CIN2;
C2G2=G7|T7&G8|T7&T8&CIN2;
C3G2=G8|T8&CIN2;
C4G2=CIN2;
C1G3=G11|T11&G12|T11&T12&G13|T11&T12&T13&CIN3;
C2G3=G12|T12&G13|T12&T13&CIN3;
C3G3=G13|T13&CIN3;
C4G3=CIN3;
C1G4=G15|T15&G16|T15&T16&G17|T15&T16&T17&CIN4;
C2G4=G16|T16&G17|T16&T17&CIN4;
C3G4=G17|T17&CIN4;
C4G4=CIN4;
C1G5=G20|T20&G21|T20&T21&G22|T20&T21&T22&CIN5;
C2G5=G21|T21&G22|T21&T22&CIN5;
C3G5=G22|T22&CIN5;
C4G5=CIN5;
C1G6=G24|T24&G25|T24&T25&G26|T24&T25&T26&CIN6;
C2G6=G25|T25&G26|T25&T26&CIN6;
C3G6=G26|T26&CIN6;
C4G6=CIN6;
C1G7=G29|T29&G30|T29&T30&G31|T29&T30&T31&CIN7;
C2G7=G30|T30&G31|T30&T31&CIN7;
C3G7=G31|T31&CIN7;
C4G7=CIN7;
C1G8=G33|T33&G34|T33&T34&G35|T33&T34&T35&CIN8;
C2G8=G34|T34&G35|T35&CIN8;

```

```

C3G8=G35|T35&CIN8;
C4G8=CIN8;
/*CALCULATE THE OUTPUT SIGNALS*/
/*FIRST GROUP*/
ARN(1)=T1&-C1G1|-T1&C1G1;
ARN(2)=T2&-C2G1|-T2&C2G1;
ARN(3)=T3&-C3G1|-T3&C3G1;
ARN(4)=T4&-C4G1|-T4&C4G1;
/* GROUP # 2*/
ARN(5)=T5&-C1G2|-T5&C1G2;
ARN(6)=T6&-C2G2|-T6&C2G2;
ARN(7)=T7&-C3G2|-T7&C3G2;
ARN(8)=T8&-C4G2|-T8&C4G2;
ARN(10)=T10&-C1G3|-T10&C1G3;
ARN(11)=T11&-C2G3|-T11&C2G3;
ARN(12)=T12&-C3G3|-T12&C3G3;
ARN(13)=T13&-C4G3|-T13&C4G3;
/*GROUP#4*/
ARN(14)=T14&-C1G4|-T14&C1G4;
ARN(15)=T15&-C2G4|-T15&C2G4;
ARN(16)=T16&-C3G4|-T16&C3G4;
ARN(17)=T17&-C4G4|-T17&C4G4;
/*GROUP#5 */
ARN(19)=T19&-C1G5|-T19&C1G5;
ARN(20)=T20&-C2G5|-T20&C2G5;
ARN(21)=T21&-C3G5|-T21&C3G5;
ARN(22)=T22&-C4G5|-T22&C4G5;
/*GROUP #6 */
ARN(23)=T23&-C1G6|-T23&C1G6;
ARN(24)=T24&-C2G6|-T24&C2G6;
ARN(25)=T25&-C3G6|-T25&C3G6;
ARN(26)=T26&-C4G6|-T26&C4G6;
/*GROUP #7 */
ARN(28)=T28&-C1G7|-T28&C1G7;
ARN(29)=T29&-C2G7|-T29&C2G7;
ARN(30)=T30&-C3G7|-T30&C3G7;
ARN(31)=T31&-C4G7|-T31&C4G7;
/*GROUP #8*/
ARN(32)=T32&-C1G8|-T32&C1G8;
ARN(33)=T33&-C2G8|-T33&C2G8;
ARN(34)=T34&-C3G8|-T34&C3G8;
ARN(35)=T35&-C4G8|-T35&C4G8;
IF ADDOUT THEN DO;
PUT SKIP(5);
CALL PRINT1(DB,'DB ');
CALL PRINT1(DR,'DR ');
CALL PRINT1(G,'G ');
CALL PRINT1(T,'T ');
CALL PRINT1(ARN,'ARN ');
PUT EDIT('GROUP','1','2','3','4','5','6','7','8')(SKIP,A(10),
8 (X(2),A(1)));
PUT EDIT('COUT(1)=',COUT1,COUT2,COUT3,COUT4,COUT5,COUT6,
COUT7,COUT8)
(SKIP,A(10),8 (X(2),B(1)));
PUT EDIT('P(1)=',P1,P2,P3,P4,P5,P6,P7,P8)
(SKIP,A(10),8 (X(2),B(1)));
PUT EDIT('CIN(1)=',CIN1,CIN2,CIN3,CIN4,CIN5,CIN6,CIN7,CIN8)

```



```

        (SKIP,A(10),8 (X(2),B(1)));
PUT EDIT('C1G(I)=' ,C1G1,C1G2,C1G3,C1G4,C1G5,C1G6,C1G7,C1G8)
        (SKIP,A(10),8 (X(2),B(1)));
PUT EDIT('C2G(I)=' ,C2G1,C2G2,C2G3,C2G4,C2G5,C2G6,C2G7,C2G8)
        (SKIP,A(10),8 (X(2),B(1)));
PUT EDIT('C3G(I)=' ,C3G1,C3G2,C3G3,C3G4,C3G5,C3G6,C3G7,C3G8)
        (SKIP,A(10),8 (X(2),B(1)));
PUT EDIT('C4G(I)=' ,C4G1,C4G2,C4G3,C4G4,C4G5,C4G6,C4G7,C4G8)
        (SKIP,A(10),8 (X(2),B(1)));

```

```

END;
END ADD32;

```

```

ADDOVF:  PROC(ADDE, AOV,      AUCG,      CSB,      CSH,
            CSW,      RSYS);
DCL(ADDE, AOV,      AUCG,      CSB,      CSH,      CSW,
    RSYS) BIT(1);
DCL ( AR, DB, DR, LR)(36) BIT(1) EXTERNAL;
PUT LIST('ADDOVF ENTERED');
/* CHECK FOR ADDER OVERFLOW */
AOV='0'B;
IF LR(9)&AUCG THEN AOV='1'B;
ELSE DO;
    IF CSW THEN AOV=ADDE&(DR(1)&DB(1)&¬AR(1)
                        |DR(1)&¬DB(1)&AR(1));
    IF CSH THEN AOV=ADDE&(DR(19)&DB(19)&¬AR(19)
                        |¬DR(19)&¬DB(19)&AR(1));

    IF CSB THEN AOV=ADDE&(DR(28)&DB(28)&¬AR(28)
                        |¬DR(28)&¬DB(28)&AR(28));
END;
IF RSYS THEN AOV='0'B;
END ADDOVF;
/*

```

## 2.8 Boolean/Shift Logic

The boolean logic accepts operands from the Distribution Register (DR) and the Distribution Bus (DB), performs one of the boolean functions, "AND", "OR", "XOR" or "EQU" on them, and gates the result into the Logic Register (LR). The "OR" function is a combination of a data transfer from the DB to the LR and from the DR to the LR.

The "XOR" and "EQU" functions are generated using parts of the 32 bit adder. The appropriate outputs are then gated from the adder to the LR.

The shift logic employs the permuter to make shifts in multiples of 8 bits. The shift control first makes the highest multiple-of-8 bit shift that it can without exceeding the desired shift. This is done using the permuter and inhibiting the necessary bytes. After this has been done the shift control shifts the DR to the LR shifting one bit at a time and then returning the LR to the DR until the proper number of additional bits have been shifted and the properly shifted result is in the LR. Since the flags are not touched in the bit shifting, the shift control only will shift flags in multiples of 8 (i.e. when the permuter is used).



## 2.8.1 Boolean/Shift Logic-Functional Description

### 2.8.1.1 Boolean Logic

As stated previously the "OR" function is a combination of a data transfer from the DB and DR to the LR. The DR and DB are simultaneously gated in complemented form to the LR "Inbus" (LRN) where they are dot-ored and then gated into the LR.

The "AND" function is realized by a NAND gate, with a DR and a DB input, in each position. The NAND outputs are dot-ored to the respective bit positions on the LRN. The LRN at this point is in complemented form but this is compensated for in the output definitions of the LR.

Exclusive or's, "XOR", and equivalences "EQV" are actually generated in the 32 bit adder. The input gates generate the function

$$\overline{T}_i = DB_i \cdot DR_i \vee \overline{DB}_i \cdot \overline{DR}_i$$

in every bit position. These  $\overline{T}_i$  signals are then routed to the boolean logic as well as to the inner portions of the adder itself. In the boolean logic, the  $\overline{T}_i$  signals are gated to the LRN inputs whenever the XOR or EQV function is selected. The function that is formed is determined as follows: If both operands are gated from the Operand Stack in 'true' form, the EQV function is formed. If one of the operands is complemented when it is gated from the OS, the XOR function is generated, since  $\overline{X} \cdot Y \vee \overline{\overline{X}} \cdot \overline{Y} = \overline{X} \cdot Y \vee X \cdot \overline{Y}$  (alternately,  $\overline{X} \oplus Y = X \oplus Y$ ).

Flag bits are treated the same way as the data bits in all boolean operations. Extra logic is employed to generate the XOR and EQV functions for the flags since none is provided in the adder.



#### 2.8.1.2 Shift Logic

Logical left and right shifts take place between the DR and the LR. Gates are provided to allow an operand in the DR to be shifted one bit position to either the right or the left while being transferred from the DR to the LR. During these shifts the positions of the flags are not changed. Shifts of more than one bit are effected by gating the LR directly back to the DR, then shifting another bit position. The TP control uses the M counter to keep track of the number of shifts still to be performed.

For shifts of 8 bits or more the Permuter Logic is also used. Before single bit shifting is performed, the DR is shifted the highest number of bytes possible without exceeding the bit shift count in the M counter. The flags are shifted along with the data bits. By using the permuter for large shifts, the one-bit-at-a-time shifter never has to make more than 7 shifts to complete a given shift command. If a shift of more than 31 positions is specified, the cell is set to all zeros.

The multiple-of-8 shift by the Permuter is made in one permutation regardless of how many byte places are being shifted. Since the permuter does not shift bytes "off the end" the shift logic must provide the proper inhibit signals to the Permuter so that these bytes will be masked out. Figure 2.8.1.2 gives a table showing how far to permute and which bytes must be inhibited as a function of the shift direction and the high-order two bits of the M counter (these bits give the number of bytes to be shifted). Using this table the following equations can be developed:

MCT <sub>1</sub>	MCT <sub>2</sub>	Left Shift		Right Shift	
		Permute Signal	Inhibit Bytes	Permute Signal	Inhibit Bytes
0	0	PL0	-	PL0	-
0	1	PL1	3	PL3	0
1	0	PL2	2,3	PL2	0,1
1	1	PL3	1,2,3	PL1	0,1,2

Figure 2.8.1.2 - Permuter Control Signals for Byte Shifting



$$PLO = LSP \cdot \bar{M}_1 \cdot \bar{M}_2 \vee RSP \cdot \bar{M}_1 \cdot \bar{M}_2$$

$$IBO = RSP \cdot (M_1 \vee M_2)$$

$$PL1 = LSP \cdot \bar{M}_1 \cdot M_2 \vee RSP \cdot M_1 \cdot M_2$$

$$IB1 = RSP \cdot M_1 \vee LSP \cdot M_1 \cdot M_2$$

$$PL2 = LSP \cdot M_1 \cdot \bar{M}_2 \vee RSP \cdot M_1 \cdot \bar{M}_2$$

$$IB2 = LSP \cdot M_1 \vee RSP \cdot M_1 \cdot M_2$$

$$PL3 = LSP \cdot M_1 \cdot M_2 \vee RSP \cdot \bar{M}_1 \cdot M_2$$

$$IB3 = LSP \cdot (M_1 \vee M_2)$$

where PLi = permuter left i bytes; RSP = right shift; LSP = left shift.

However, referring to the equations for PL1 and IB2, and PL3 and IB1 some simplification can take place if we let

$$IBPL21 = RSP \cdot M_1 \cdot M_2$$

$$IBPL13 = LSP \cdot M_1 \cdot M_2$$

where IBPLij = is a control line to the permuter which inhibits byte i and also permutes left j bytes.

These signals go directly to the Permutation Control Logic and turn on the appropriate permute and inhibit signals. This means that the remaining parts of the shift equations can be written as follows:

$$PLO = LSP \cdot \bar{M}_1 \cdot \bar{M}_2 \vee RSP \cdot \bar{M}_1 \cdot \bar{M}_2$$

$$IBO = RSP \cdot (M_1 \vee M_2)$$

$$PL1 = LSP \cdot \bar{M}_1 \cdot M_2$$

$$IB1 = RSP \cdot M_1$$

$$PL2 = LSP \cdot M_1 \cdot \bar{M}_2 \vee RSP \cdot M_1 \cdot \bar{M}_2$$

$$IB2 = LSP \cdot M_1$$

$$PL3 = RSP \cdot \bar{M}_1 \cdot M_2$$

$$IB3 = LSP \cdot (M_1 \vee M_2)$$

Using these equations the permutation and inhibit signals are decoded from the M-counter and the inputs from the TP control and are then routed to the permuter. The rest state of the permuter is disabled during this cycle when the operand is being shifted in steps of 8.

Double words are not shifted, primarily because of the problems involved in masking bits shifted out of one half of the double word into the proper positions in the other half.

When bytes or half-words are to be shifted they are initially loaded into the DR by the Permuter, right or left justified depending on the direction of the shift. The cell is placed so that the bits will be shifted "off the end" of the DR and not into the "unused" portion. During these loadings the "unused" portion of the DR is set to zero by inhibiting the proper permuter bytes.

The Shift Logic contains several sections of "testing" logic to detect certain conditions of which the TP control must be aware. MZER is "one" if the M counter and all the high order bits of the shift count (which is in the right-most bytes of the IR) are zero.

## 2.8.2 Signal Name Lists for the Boolean/Shift Logic

### 2.8.2.1 Control Signals

AND/E/	-	AND enable DB • DR → LR
BLR/G/	-	Gate DB → LR
CTD/E/	-	Enable count down for M counter (MCT)
CTMHZ	-	Output Signal - High Order 2 bits of the MCT = 0
CTMLZ	-	Output Signal - Low Order 3 bits of the MCT = 0
CTU/E/	-	Enable count up for M counter (MCT)
DLR/G/	-	Gate DR → LR
EQV/E/	-	Enable equivalence
LS/E/	-	Enable - left shift
LSP/E	-	"left shift" to be made by shift control using permuter
MCT/E/	-	Enable M-counter
MZER	-	Output - M-counter and high order bits of IR all = 0.
ØR/E/	-	ØR enable DB v DR → LR
RMCT/	-	Reset M-counter
RS/E/	-	Enable - right shift
RSP/E	-	"right shift" to be made by shift control using permuter
XØR/E/	-	Exclusive ØR enable DB ⊕ DR → LR



### 2.8.2.2 Internal Signals Used by the Boolean/Shift Logic

CTD	-	When on, M counter counts down
CTU	-	When on, M counter counts up
DBi	-	Distribution bus - bit i
DRI	-	Distribution register - one of the inputs for shift - bit i
IBi/	-	Inhibit signal for byte i in permuter output
IBPL13	-	Inhibit byte 1 - permute left 3
IBPL21	-	Inhibit byte 2 - permute left 1
IRi	-	Instruction Register - bit i
LRi	-	Logic register - output of shift/boolean - bit i
LRi/G	-	LR input gating signal: LRN → LR - byte i
LRLi	-	LR latch, must be 0 for LRN → LR - byte i
LRNi	-	Input bus to logic register - bit i
MCTi	-	M counter - bit i
PLi/	-	Permute left i bytes
Ti/	-	Adder Output - equals EQV of DR and DB - bit i
XRV/E	-	Exclusive or or equivalence enable (goes to permuter)



## 2.8.3 Boolean/Shift Logic - PL/1 Description

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
      HNDLSHFT: PROC(
          ANDE,      CTDE,      CTMHZ,      CTMLZ,      CTUE,      DLRG,
          IBPL13,    IBPL21,    LRMCTG,    LSE,        LSPE,      MCTE,
          ORE,       RMCT,      RSE,        RSPE,
          IB,        PL);
      DCL ( ANDE,CTDE,      CTMHZ,      CTMLZ,      CTUE,      DLRG,
            IBPL13,    IBPL21,    LRMCTG,    LSE,        LSPE,      MCTE,
            MCTE,      ORE,       RMCT,      RSE,        RSPE) BIT(1),
            (IB, PL)(0:3) BIT(1);
      DCL (DB,DR,LR,LRN)(36) BIT(1) EXTERNAL;
      DCL (CTU,CTD) STATIC BIT(1), (CTLZ,CTHZ,CTZ)BIT(1),
            MCT(1:5) BIT(1) STATIC;
      LRN='0'B;
          /* DRAWING 07-1*/
      IF RSE THEN DO;
          DO I=0 TO 3;
              DO J=I*9+1 TO I*9+8;
                  LRN(J+1)=LRN(J+1)|DR(J);
              END;
          END;
          DO I=9 TO 36 BY 9;
              LRN(I)=DR(I);
          END;
          LRN(1)='0'B;
      END;
      IF LSF THEN DO;
          DO I=0 TO 3;
              DO J=I*9+1 TO I*9+8;
                  LRN(J-1)=LRN(J-1)|DR(J);
              END;
          END;
          DO I=9 TO 36 BY 9;
              LRN(I)=DR(I);
          END;
          LRN(35)='0'B;
          CALL PRINT1(LRN,'LRN ');
      END;
      IF ANDE THEN LRN(*)=DR(*)&DB(*);
      /* CALCULATE CONTROL INFORMATION DRAWING 17-1 */
      CTMHZ=~MCT(5)&~MCT(4);
      CTMLZ=~MCT(3)&~MCT(2)&~MCT(1);
      /* LOAD M-COUNTER FROM THE LR DRAWING 17-1 */
      IF LRMCTG THEN DO I=1 TO 5;
          MCT(I)=LR(I+30);
      END;
      /*RESET COUNTER,SELECT COUNT UP OR DOWN, AND COUNT */
          /* DRAWING 17-2 */

      IF RMCT THEN MCT='0'B;
      IF CTUE THEN DO;
          CTU='1'B;
          CTD='0'B;
      END;
```

```

IF CTDE THEN DO;
    CTD='1'B;
    CTU='0'B;
END;
IF MCTE THEN CALL MCTR;

```

```

/* DETERMINE BYTE SHIFTING FOR PERMUTER */
/* DRAWING 17-1*/

```

```

IBPL21=RSPE&MCT(1)&MCT(2);
IBPL13=LSPE&MCT(1)&MCT(2);
PL(0)=¬MCT(1)&¬MCT(2)&(LSPE|RSPE);
PL(1)=LSPE&¬MCT(1)&MCT(2);
PL(2)=(MCT(1)&¬MCT(2))&(LSPE|RSPE);
PL(3)=RSPE&¬MCT(1)&MCT(2);
IB(0)=RSPE&(MCT(1)|MCT(2));
IB(1)=RSPE&MCT(1);
IB(2)=LSPE&MCT(1);
IB(3)=LSPE&(MCT(1)|MCT(2));

```

```

DLRG=DLRG|ORE;
MCTR:PROC;
    DCL (M,I) FIXED BIN (5,0), MS BIT(5), ONE FIXED BIN;
    PUT LIST('MCTR ENTERED');
    ONE=1;
    DO I=1 TO 5;
        IF MCT(I) THEN M=M+2**(5-I);
    END;
    IF CTU THEN M=M+1;
    IF CTD THEN M=M-1;
    IF M<0 THEN MCT='1'B;
    ELSE DO;
        MS=M;
        DO I=1 TO 5;
            MCT(I)=SUBSTR(MS,I,ONE);
        END;
    END;
END MCTR;

END BOOLSHFT;

```

```

/*

```



## 2.8.4 Boolean/Shift Logic - Logical Description

### 2.8.4.1 Basic Boolean/Shift Logic

The basic Boolean/Shift Logic is used to perform the actual boolean and shifting operations on data. This logic is really quite straight forward. It consists of various sets of gates each of which can be operated separately. Each set has 32 bit positions and the corresponding bit positions of each set are dot-ored together to form a 32 bit input bus ( $\overline{\text{LRNi}}$ ) to the Logic Register (LR). The flag positions of the input data are handled separately and are also gated to their corresponding positions in the LRNi.

The various boolean operations are accomplished as explained in Section 2.8.1.1 and are activated by turning on the necessary gate signals, i.e. XOR/E, EQV/E, AND/E, and OR/E. The shifting operations are performed between the DR and the LR by using a simple gate, but gating each input bit position to the corresponding output bit position which is one place to the right or left. In the case of a right or left shift by one bit, the flags are gated straight through from the DR to the LR without changing position.



#### 2.8.4.2 M-Counter Logic

The M-Counter, as shown in the TP Logic Book, drawing 17-2, is a double rank, up-down counter with a combination of ripple and lookahead carry logic. The information is stored in the upper rank and can be retrieved in true or complement form. The choice of up or down counting is made by setting the direction flip-flop to either CTU or CTD by means of the  $\overline{\text{CTU/E}}$  or  $\overline{\text{CTD/E}}$  control signals, respectively.

As in a normal double rank counter, the upper rank, which contains the current state of the counter, is gated into the lower rank of the counter during the time that the M Counter is not counting (i.e.  $\overline{\text{MCT/E}} = 1$ ). The output of the lower rank, in turn, is sent to the carry propagation logic which in turn decides what the state of the M-Counter will be the next time that  $\overline{\text{MCT/E}}$  is activated. However, since  $\overline{\text{MCT/E}}$  is not active at this time, this result sits at the input of the upper rank until such time as MCT/E goes to 0 (i.e.  $\text{MCT/E} = 1$ ). As soon as this happens, the gates to the lower rank close. This ensures that the inputs (and thus the outputs also) of the carry propagation logic will not change. One collector delay later, the output of the carry propagation logic is gated into the upper rank. Thus, provided that there is a sufficient length of time for the carry propagation logic to have settled down after the previous count, a new count value will appear at the output of the M-Counter 1 collector delay time plus 1, 260 flip-flop storage time after  $\overline{\text{MCT/E}}$  goes to 0.

The most complicated part of the M-Counter logic is the carry propagation logic. It is also the key to the whole M-Counter operation. Its inputs consist of the 5 data signals from the lower rank of the counter (in both true and complement form), the CTU and CTD signals which determine whether we will add 1 or subtract 1, and the  $\overline{\text{MCT/E}}$  signal which is inverted and then used to gate the output of the carry propagation logic into the M-Counter's upper rank.

As mentioned earlier, the carry propagation logic uses both ripple and lookahead carry propagation techniques. The lookahead is used between bits 1 and 2 and between bits 3, 4, and 5 where bit 1 is

the high order bit and bit 5 is the low order bit. A ripple carry is performed between bits 2 and 3.

The carry propagation logic itself can be thought of as two sets of gates which if both are turned on, cause the corresponding bit of the counter to change state. A simplified logic diagram is shown in Figure 2.8.4.2/1. The second gate is activated when the count enable pulse is activated while the first gate is activated by a carry (or borrow) from the previous lower order stage. Note that the inputs to the gate system are arranged so that if either gate is off, the bit represented by the upper rank flip-flop will not change. As can be seen, therefore, the new state will be determined by what appears on the second gate line at each position of the counter.

In actual fact, the "carry gates" at each counter position are determined by 2 NAND circuits which are dot-ored together, one NAND being activated when the M-Counter is counting up (CTU = 1) and the other NAND being activated when the M-Counter is counting down (CTD = 1). The single exception to this scheme is the low order bit position, MCT5, which always changes state when the MCT/E signal goes to 0 regardless of whether CTU or CTD is 1.

As an example, in the case of a count up, MCT<sup>4</sup> will change state only if the next lower order position, MCT<sup>5</sup>, was a 1 during the previous counter state. This is due to the fact that if a 1 is added to the MCT<sup>5</sup> position when it is already 1, a carry into position MCT<sup>4</sup> will result which in turn will cause MCT<sup>4</sup> to change state. On the other hand in the case of a count down, MCT<sup>4</sup> will change state only if the next lower order position, MCT<sup>5</sup>, was a 0 during the previous counter state. This is due to the fact that if a 1 is subtracted from the MCT<sup>5</sup> position when it is in the 0 state, it will have to borrow from the MCT<sup>4</sup> position which in turn causes MCT<sup>4</sup> to change state. The logic used to generate the "carry gate" for position 4 is shown in Figure 2.8.4.2/2. Note that in Drawing 17-2 of the TP Logic Book the MCT<sup>5</sup> and  $\overline{\text{MCT}}^5$  signals are actually taken from the lower rank flip-flops of the M-Counter.

For the MCT<sup>3</sup> position things become slightly more complicated. In the first place the conditions which determine whether or not MCT<sup>3</sup> changes state depend on both of the two lower order positions, MCT<sup>4</sup> and MCT<sup>5</sup>. In a count up they must both be 1 and in a count down they must

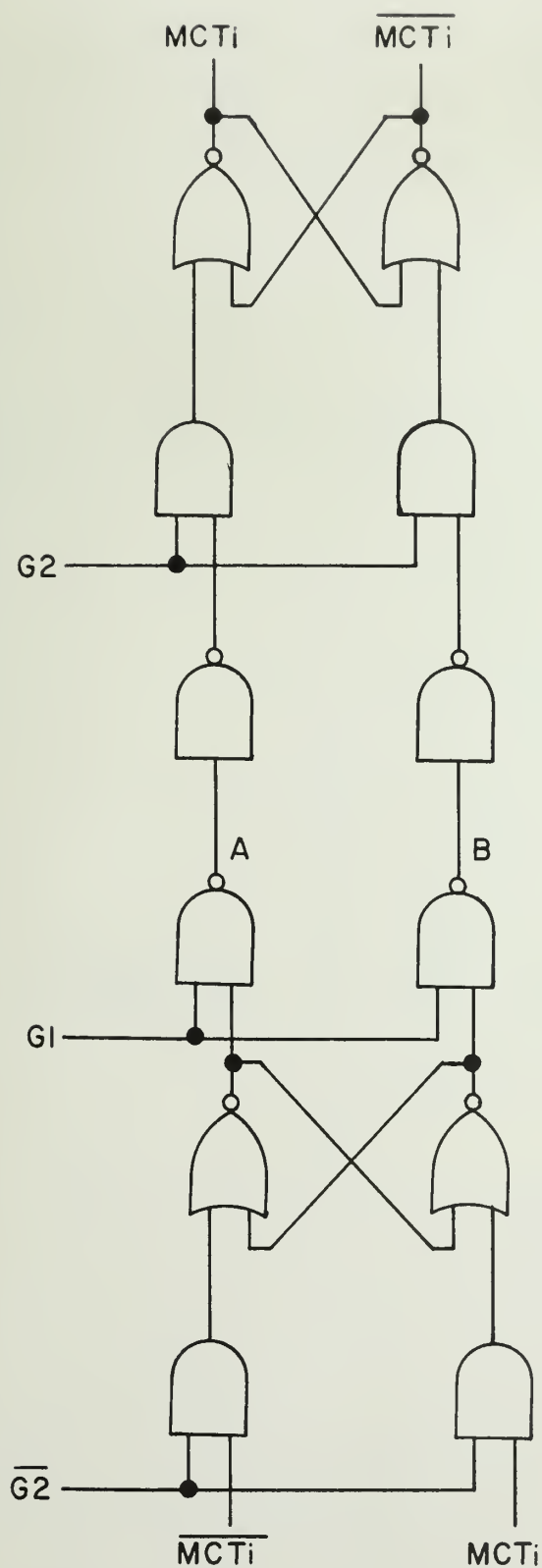


Figure 2.8.4.2/1 - Simplified Logic for One Position of the M-Counter

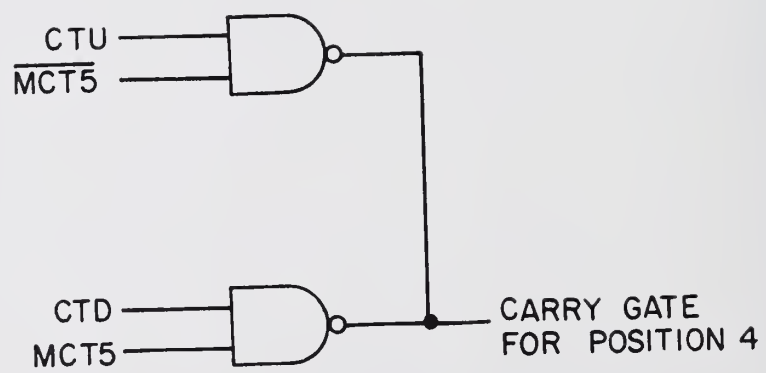


Figure 2.8.4.2/2 - Carry Gate Logic for Position MCT<sup>4</sup>

both be 0 in order to produce either a carry into or a borrow from the third position. In Figure 2.8.4.2/3 the logic used to generate the 3rd position "carry gate" is shown. Note that there are actually 2 "carry gates", one for each previous position, and that both must be on in order to cause a change of state. Each of these "carry gates" is produced in the same way as the "carry gate" for position MCT<sup>4</sup>. In fact one of them is actually the same signal as was used in the MCT<sup>4</sup> position.

It can be seen from the above description, that if we wanted to use full lookahead, each higher order position of the counter could be constructed by adding an additional "carry gate" and AND'ing it along with all of the previous gates. However this method would create the need for NAND circuits with more and more inputs and since there is no need for the high speed which can be achieved with a full lookahead counter, it was decided to save logic by using a ripple carry between the 2nd and 3rd positions.

Using a ripple carry simply means that in the 2nd M-Counter position, instead of looking at the contents of all 3 lower order positions we will simply sit around and wait until the previous logic has decided whether or not the immediately preceding position, MCT<sup>3</sup>, will have to change state. It is called a ripple carry because if every counter position acted this way we could only find out the new count one bit position at a time beginning at the low order end of the counter, and the new count would "ripple" through to the high order end one bit position at a time.

If it turns out that the MCT<sup>3</sup> position does change state then one of the outputs from the "carry gate" circuits (labelled A and B in Figure 2.8.4.2/1) will be '1' and the other will be '0'. B will be '1' if MCT<sup>3</sup> is going from 1 to 0 and A will be '1' if MCT<sup>3</sup> is going from 0 to 1. If no change in MCT<sup>3</sup> is to occur, both A and B will be 0. Thus the generation of the carry gate for the MCT<sup>2</sup> position can be accomplished by the logic shown in Figure 2.8.4.2/4.

Finally the two carry gates for M-Counter position 1 can be generated exactly the same way as for position 3 except that this time the results from position 2 and the ripple results from position 3 are used.



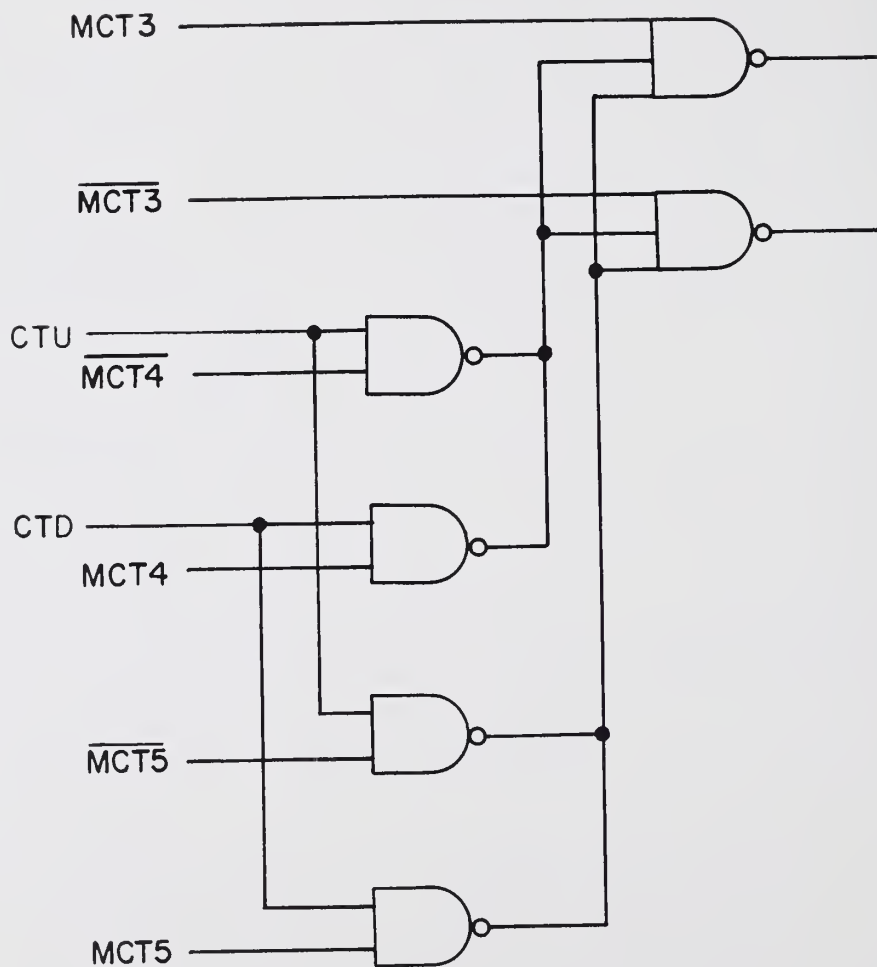
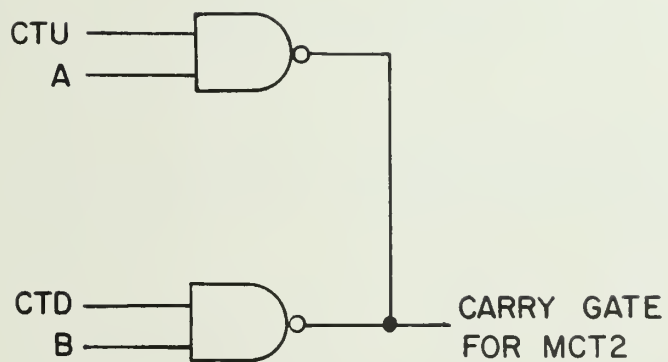


Figure 2.8.4.2/3 - Carry Gate Logic for Position MCT3





Note: the A and B inputs come from the positions indicated in Figure 2.8.4.2/1.

Figure 2.8.4.2/4 - Carry Gate Logic for "Ripple Carry" to Position MCT2



## 2.9 Algebraic/Logical Compare Logic

The algebraic/logical compare logic in the TP is used to make comparisons between byte, halfword or word size cells. If it is necessary to compare two floating point or BCD numbers, this is done using the Arithmetic Unit.

Generally speaking the comparisons are made by subtracting and then comparing the result with zero. Therefore the hardware is set up to test the AR for zero. Various groups of bits are tested depending on the cell size and the type of comparison. The sign is also checked to set the "greater" and "less" flip-flops.

The compare is usually done in two cycles. During the first the "greater" or "less" flip-flops are set according to the signs and the EQ is set to 1 if the AR is zero. In the second cycle, both the "greater" and "less" flip-flops are reset to zero if the EQ flip-flop was previously set to one.

In the following section the various hardware descriptions are given. The actual sequencing of the various kinds of comparisons is given in Section 4.3.2.1.



### 2.9.1 A/L Compare Logic - Functional Description

The A/L Compare Logic is used to perform algebraic and logical comparisons between two numbers. In an algebraic compare, the numbers are treated as 2's complement numbers with the highest order bit being the sign bit. In a logical compare, the numbers are treated as unsigned positive numbers.

The logic for the A/L compare is shown in the 05-series of drawings in the TP Logic Book. In order to understand this logic, it is necessary to know how the control logic uses it. The basic comparison method, as previously mentioned in Section 2.9, uses a subtraction. In certain cases this is not necessary since the results of the comparison can be determined directly from the high order bits of the two numbers when these are different. When the sign bits are the same, however, the subtraction must be performed and the proper indicator settings determined from the sign bit of the result.

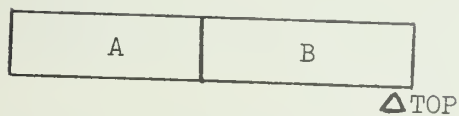
Since the cells are right justified and may be either bytes, half-words, or words, the actual sign positions which are used will depend on the cell size. Only the position corresponding to the highest order bit of the present cell size cell is used.

Figure 2.9.1/1 shows a table giving the algebraic and logical interpretation of the various bit strings in the set of 4-bit binary numbers. Note that if two numbers are interpreted as algebraic they can always be compared by 2's complement subtraction and observation of the sign of the result. This will also hold for logical numbers if both numbers have the same first bit. It will not work if the numbers have opposite high order bits. However, it is exactly this situation which can be predicted without subtraction. In the case of logical numbers the operand with the '1' in the first bit position is larger while in the case of algebraic numbers the operand with the '0' in the first bit is larger. Figure 2.9.1/2 gives the indicator settings for the situation in which both operands have the same high order bit.

Algebraic Interpretation	Bit String	Logical Interpretation
-1	1111	15
-2	1110	14
-3	1101	13
-4	1100	12
-5	1011	11
-6	1010	10
-7	1001	9
-8	1000	8
7	0111	7
6	0110	6
5	0101	5
4	0100	4
3	0011	3
2	0010	2
1	0001	1
0	0000	0

Figure 2.9.1/1 - Algebraic and Logical Interpretation  
of 4-Bit Numbers

Operand Stack:



Compare: A  $\begin{matrix} > \\ < \end{matrix}$  B

First gate B  $\rightarrow$  DR (true); gate A  $\rightarrow$  DB (complemented)

Second Form B - A; gate result into AR

Result in AR	A	B	Compare Alg.	Compare Log
AR >0 (+)	small +	large +	<	<
>0 (+)	large -	small -	<	<
=0 (+)		(B=A)	=	=
<0 (-)	large +	small +	>	>
<0 (-)	small -	large -	>	>

Figure 2.9.1/2 - Setting the GT and LT Flip-Flops When the First Bit of Both Operands is the Same

The 05-2 drawing shows the equality compare circuit. After the subtraction this circuit checks for zero in the AR. Note that the cell size signal is used to control which bits are checked. If all of the applicable bits are zero, the EQ flip-flop is set.

Note that even if the EQ flip-flop is set, one of the GT or LT flip-flops will still be set on the basis of the contents of the AR and DR sign positions. Therefore in this case the RGL/E signal can be turned on and if EQ is on, the GT and LT flip-flops will be reset.

The flag match logic is shown in Drawing 05-3. It is very simple and merely compares the flags on the DR and DB before the subtraction takes place.

The match checking is effected by a gated "equivalence" between the flags of the top and next-to-top operands in the OS. However, since the second from the top operand was gated out in complement form, an "exclusive or" is performed rather than an equivalence; the results are the same, though, since:

$$FM = A \oplus B = A B \vee \bar{A} \bar{B} \text{ and if } \bar{C} \text{ is substituted for } A:$$

$$FM = \bar{C} B \vee C \bar{B}$$

If any pair of flags do not match, the FM indicator is not set. The final step for the main control in these instruction sequences is to "pop" the top cell out of the OS.



## 2.9.2 Signal Name Lists for the A/L Compare Logic

### 2.9.2.1 Control Signals

AUC/G	-	Gate AU condition code
CPA/E/	-	Algebraic compare enable
CPH/E/	-	Hollerith compare enable
CPL/E/	-	Logical compare enable
FLM/E/	-	Flag match enable
RGL/E	-	Reset GT and LT if EQ = 1
RSYS/	-	Reset system
TZ/E/	-	Test zero enable



#### 2.9.2.2 Internal Signals Used by the A/L Compare Logic

ARi	-	Arithmetic Register - bit i
CSB	-	Cell size is byte
CSH	-	Cell size is halfword
CSW	-	Cell size is word
DBi	-	Distribution bus, from the permuter - bit i
DRi	-	Distribution Register
EQ	-	Equality flip-flop
FM	-	Flags of cells in DR and DB match
GT	-	Greater than flip-flop
LT	-	Less than flip-flop



# 2.9.3 A/L Compare Logic - PL/1 Description

```

/ EXFC PL1
/PL1.SYSPUNCH DD SYSOUT=B
/PL1.SYSIN DD *
  COMPARE:  PROC(      AUCG,      CPAE,      CPHE,      CPLE,
                    CSB,      CSH,      CSW,      EQ,      GT,
                    LT,      RCOMFF,  REQS,  RGLF,  SGTS,SLTS,
                    TZE);
  DCL(AUCG, CPAE,      CPHE,      CPLF,      CSB,      CSH,
        CSW,      EQ,      GT,      LT,      RCOMFF,
        REQS,RGLE,SGTS,SLTS,TZE)  BIT(1);
  DCL( AR,DR,LR)(36) BIT(1) EXTERNAL;
  DCL (I,ZERO)FIXED BIN ;
  IF RCOMFF THEN LT,GT='0'B;
  IF EQ&RGLE THEN GT='1'B;
  IF LR(27)&AUCG|SLTS THEN LT='1'B;
  IF DR(9)&AUCG|SGTS THEN GT='1'B;
  IF (AR(28)&CSB&CPLE
      |AR(1)&CSW&(CPLE|CPAE)
      |AR(19)&CSH&(CPLE|CPAE)
      |AR(35)&CPHE)
      THEN GT='1'B;

  IF (¬AR(28)&CSB&CPLE
      |¬AR(1)&(CPLE|CPAE)&CSW
      |¬AR(19)&(CPLE|CPAE)&CSH
      |¬AR(35)&CPHE)
      THEN LT= '1'B;

  IF RCOMFF|REQS THEN EQ='1'B;
  IF ¬LR(36)&AUCG THEN EQ='1'B;
  IF (CPLE|CPAE|TZE) THEN DO;
    ZERO='0'B;
    IF (CSB|CSH|CSW) THEN DO I=28 TO 35;
      ZERO=ZERO|AR(I);
    END;
    IF (CSH|CSW) THEN DO I=19 TO 26;
      ZERO=ZERO|AR(I);
    END;
    IF CSW THEN DO I=1 TO 17 WHILE (I¬=9);
      ZERO=ZERO|AR(I);
    END;
    IF ¬ZERO THEN EQ='1'B;
  END;
END COMPARE;

```

11/5/70

Section 2.9.3 - 1/1



## 2.10 Cell Size Generator

The cell size generator is used to drive the cell size signals for the various logic and control groups. These output signals consist of true and complement forms of CSB, CSH, CSW and CSD which indicate byte, halfword, word and double word cell sizes, and CSBH which indicates that the cell size is either a byte or a halfword. Only one of the first four signals will be active at any one time and it is possible that none of them may be on.

There are four possible determinants for the cell size to be activated:

- 1) the field designator bits when interpreted as a cell size or immediate address field selector (selector state 00),
- 2) the field designator bits when interpreted as a number type (selector state 01),
- 3) the contents of the control cell size flip-flops (selector state 10), or
- 4) the four individual cell size control signals (selector state 11).

At any given time the decoding method actually chosen is determined by a four state selector. If the control cell size signals are chosen, and no control cell size signal is turned on, none of the signals will be activated. At most only one of the control cell size signals will be on at any one time.

In addition to the cell size signals, this block of logic also produces the signals which indicate the number type for arithmetic instructions. The rest of this section gives a more detailed explanation of the various parts of the cell size generator.





### 2.10.1 Cell Size Generator-Functional Description

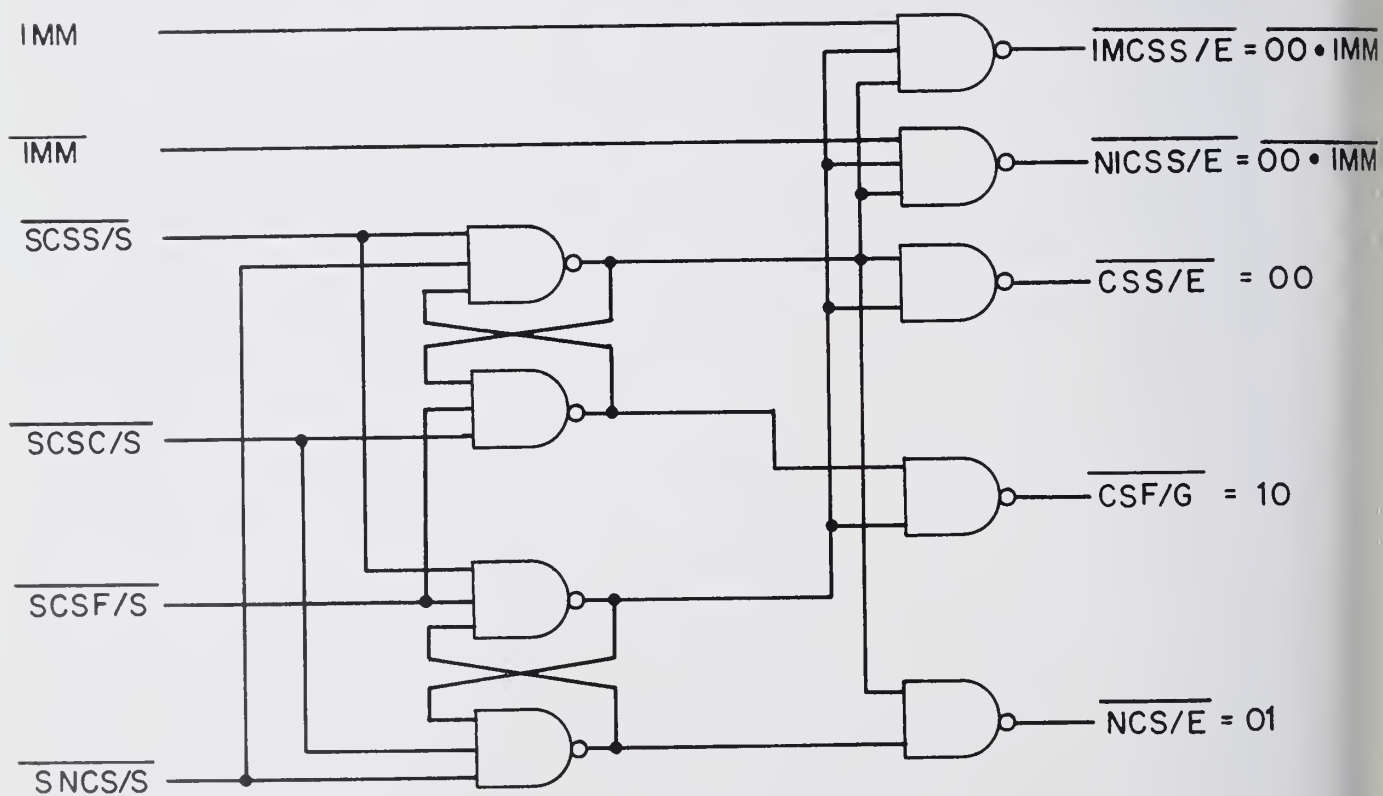
The main output bus of the Cell Size Generator consists of five signals obtained by dot-or'ing the outputs of several input gating circuits. Each gating circuit transmits data from a different source which is decoded to activate the proper signal(s). The gate signals for the gating circuits are generated by means of decoding logic attached to a selector (see Figure 2.10.1). Two flip-flops are set to one of four possible states by four different control signals. The state is then decoded by a group of NAND's. If the fourth state is set (11), all of the gates are turned off.

Note that the 00 state of the selector is actually used to activate three possible signals,  $\overline{\text{IMCSS/E}}$ ,  $\overline{\text{NICSS/E}}$ , and  $\overline{\text{CSS/E}}$ . These three signals distinguish two possible decodings of the IR7 and IR8 bit positions which contain the field designator code. In the 00 state, if there is an immediate address option for the instruction and this option is used, then  $\text{IMM} = 1$ ,  $\overline{\text{IMCSS/E}} = 0$  and the following decoding is used:

IR7	IR8	Cell Size	Immediate Field
0	0	halfword	value field
0	1	halfword	link field
1	0	word	link-value field
1	1	halfword	segment name field

If  $\text{IMM} = 0$  then  $\overline{\text{NICSS/E}} = 0$  and the decoding is strictly according to cell size and is as follows:

IR7	IR8	Cell Size
0	0	byte
0	1	halfword
1	0	word
1	1	double word



NOTE: THE 11 STATE CAUSES ALL GATES TO BE INACTIVE

FIGURE 2.10.1  
CELL SIZE GENERATOR GATE SELECTOR

Note that 01 and 10 codes are the same whether IMM is on or off. Therefore these decoders are driven by the  $\overline{\text{CSS/E}}$  signal.  $\overline{\text{IMCSS/E}}$  and  $\overline{\text{NICSS/E}}$  are used to drive the proper cell size signals when IMM = 1 and IMM = 0, respectively.

If the NCS/E signal is turned on, (i.e. the selector state is 01) the cell size signals are determined by the IR7 and IR8 bits which contain a field designator indicating a number type. One of the number type signals will also be turned on in this case. The code is as follows:

IR7	IR8	Number Type	Cell Size
0	0	short fixed	halfword
0	1	long fixed	word
1	0	floating point	doubleword
1	1	hollerith	doubleword

If the CSF/G signal is turned on, (i.e. the selector state is 10) the cell size signals are determined by the control cell size flip-flops. The coding is the same as that used for the cell size field designator case (IMM=0) except that two control flip-flops take the place of IR7 and IR8. These flip-flops are set in a manner similar to the selector used in the gating signals.

If the selector is in the fourth state where the three gating signals are off, the cell size signals may be set by using one of the control cell size signals CCSB/E, CCSH/E, CCSW/E, or CCSD/E which will activate the proper cell size signals.



## 2.10.2 Signal Name Lists for Cell Size Generator

### 2.10.2.1 Control Signals

CCSB/E	-	Enable byte output line
CCSD/E	-	Enable double word output line
CCSH/E	-	Enable halfword output line
CCSW/E	-	Enable word output line
CSBF/E	-	Set cell size control flip-flop to byte
CSDF/E	-	Set cell size control flip-flop to double word
CSF/G	-	Gate control flip-flops to CS generator
CSHF/E	-	Set cell size control flip-flop to halfword
CSS/E	-	Enable decoding of field designator bits as a cell size independent of immediate option
CSWF/E	-	Set cell size control flip-flop to word
IMCSS/E/	-	Immediate operand - cell size option
IMM	-	Control FF indicates instruction uses immediate option
NCS/E	-	Enable decoding of field designator bits as a number type
NICSS/E/	-	No immediate operand - cell size option
SCSC/S	-	Set CS gate selector off for control signal select
SCSF/S	-	Set CS gate selector for CSF/G
SCSS/S	-	Set CS gate selector for CSS/E
SNCS/S	-	Set CS gate selector for NCS/E



#### 2.10.2.2 Internal Signals Used by Cell Size Generator

CSB	-	Cell size is byte
CSBH	-	Cell size is byte or halfword
CSD	-	Cell size is double word
CSH	-	Cell size is halfword
CSW	-	Cell size is word
FPT/	-	Floating point (CSD)
HØL/	-	Hollerith (CSD)
IRi	-	Instruction Register, bit i
LFX/	-	Long fixed (CSW)
SFX/	-	Short fixed (CSH)





### 2.10.3 Cell Size Generator - PL/1 Description

```

/ EXEC PL1
/PL1.SYSPUNCH DD SYSOUT=B
/PL1.SYSIN DD *
  CSGEN: PROC(   CCSBE,   CCSDE,   CCSHE,   CCSWE,   CSB,
                CSBFE,   CSBH,    CSD,     CSDFE,   CSFG,
                CSH,     CSHFE,   CSSE,    CSW,     CSWFE,
                FPT,     HOL,     IMM,     LFX,     NCSE,
                SCSCS,   SCSFS,   SCSSS,   SFX,     SNCSS);
  DCL(CCSBE,CCSDE,   CCSHE,   CCSWE,   CSB,   CSBFE,
       CSBH,   CSD,   CSDFE,   CSFG,   CSH,
       CSHFE,   CSSE,   CSW,   CSWFE,   FPT,
       HOL,     IMM,   LFX,   NCSE,   SCSCS,
       SCSFS,   SCSSS,   SFX,   SNCSS) BIT(1);
  DCL IR(36) BIT(1) EXTERNAL;
  DCL (GSFF,      /* GATE SELECTOR FLIP-FLOP OUTPUT */
       CSCFF)    /* CELL SIZE CONTROL FLIP-FLOP OUTPUT */
        (1:2) BIT(1) STATIC,
        (IMCSSE, /* CELL SIZE DETERMINED BY IMM OPTION */
         NICSSE) /* CELL SIZE DETERMINED WITH IMM OPTION OFF */
        BIT(1);

  /*RESFT CELL SIZE SIGNALS TO ZERO*/
  CSB,CSBH,CSH,CSW,CSD ='0'B;
SET_SELECTOR:
  /* SET CELL SIZE GATE SELECTOR FLIP-FLOP STATES */

  /* SET GATE SELECTOR FOR CSSE */

  IF SCSSS THEN DO;
    GSFF(1)='0'B;
    GSFF(2)='0'B;
  END;

  /* SET GATE SELECTOR FOR NCSE */

  IF SNCSS THEN DO;
    GSFF(1)='0'B;
    GSFF(2)='1'B;
  END;

  /* SET GATE SELECTOR FOR CSFG */

  IF SCSFS THEN DO;
    GSFF(1)='1'B;
    GSFF(2)='0'B;
  END;

  /* SET GATE SELECTOR OFF FOR CONTROL SIGNAL SELECT */

  IF SCSCS THEN DO;
    GSFF(1)='1'B;
    GSFF(2)='1'B;
  END;

```

```
DECODE_SFLECTOR:  
/* CHECK IF FIELD DESIGNATOR BITS ARE TO BE DECODED AS A CELL  
SIZE INDEPENDENT OF IMMEDIATE OPTION */
```

```
CSSE=~GSFF(1)&~GSFF(2);  
NICSSE=CSSE&~IMM;  
IMCSSE=CSSE&IMM;
```

```
/* CHECK IF CONTROL FLIP-FLOPS ARE TO BE GATED TO CFL  
GENERATOR */
```

```
CSFG=GSFF(1)&~GSFF(2);
```

```
/*CHECK IF FIELD DESIGNATOR BITS ARE TO BE DECODED AS A  
NUMBER TYPE */
```

```
NCSE=~GSFF(1)&GSFF(2);
```

```
SET_CSCFF:  
/*SET CELL SIZE CONTROL FIIP-FLOP STATES */
```

```
/* ENABLE BYTE OUTPUT LINE */
```

```
IF CSBFE THEN DO;  
CSCFF(1)='0'B;  
CSCFF(2)='0'B;  
END;
```

```
/* ENABLE HALFWORD OUTPUT LINE */
```

```
IF CSHFE THEN DO;  
CSCFF(1)='0'B;  
CSCFF(2)='1'B;  
END;
```

```
/* ENABLE WORD OUTPUT LINE */
```

```
IF CSWFE THEN DO;  
CSCFF(1)='1'B;  
CSCFF(2)='0'B;  
END;
```

```
/* ENABLE DOUBLE WORD OUTPUT LINE */
```

```
IF CSDFE THEN DO;  
CSCFF(1)='1'B;  
CSCFF(2)='1'B;  
END;
```

SET\_CELL\_SIZES:

/\* SET CELL SIZES \*/

/\* CHECK IF CELL SIZE IS TO BE DETERMINED INDEPENDENTLY  
OF THE IMMEDIATE OPTION \*/

IF CSSE THEN DO;  
    CSH= $\neg$ IR(7)&IR(8);  
    CSBH= $\neg$ IR(7);  
    CSW=IR(7)& $\neg$ IR(8);  
END;

/\* CHECK IF CELL SIZE TO BE DETERMINED BY THE IMMEDIATE  
OPTION ON \*/

IF IMCSSE THEN DO;  
    CSH= $\neg$ IR(7)& $\neg$ IR(8)|CSH;  
    CSW= IR(7)& IR(8)|CSW;  
END;

/\* CHECK IF CELL SIZE IS DETERMINED BY IMMEDIATE OPTION OFF\*/

IF NICSSE THEN DO;  
    CSB= $\neg$ IR(7)& $\neg$ IR(8)|CSB;  
    CSD= IR(7)& IR(8)|CSD;  
END;

/\* CHECK IF CELL SIZE IS TO BE DETERMINED BY CELL SIZE CONTROL  
FLIP-FLOPS \*/

IF CSFG THEN DO;  
    CSB= $\neg$ CSCFF(1)& $\neg$ CSCFF(2)|CSB;  
    CSH= $\neg$ CSCFF(1)& CSCFF(2)|CSH;  
    CSBH= CSCFF(1)|CSBH;  
    CSW= CSCFF(1)& $\neg$ CSCFF(2)|CSW;  
    CSD= CSCFF(1)& CSCFF(2)|CSD;  
END;

/\* CHECK IF FIELD DESIGNATOR BITS ARE TO BE DECODED AS A  
NUMBER TYPE \*/

IF NCSE THEN DO ;  
    CSH= $\neg$ IR(7)&IR(8)|CSH;  
    CSBH= $\neg$ IR(7)& $\neg$ IR(8)|CSBH;  
    CSD= IR(7)|CSD;  
  
    FPT= IR(7)& $\neg$ IR(8);  
    HOL= IR(7)& IR(8);  
    SFX= $\neg$ IR(7)& $\neg$ IR(8);  
    LFX= $\neg$ IR(7)& IR(8);  
END;  
ELSE FPT,HOL,SFX,LFX='0'B;

/\* CHECK IF CELL SIZE IS DETERMINED BY CONTROL \*/

CSB=CSB|CCSBE;  
CSH=CSH|CCSHE;  
CSBH=CSBH|CCSHE|CCSBE;  
CSW=CSW|CCSWE;  
CSD=CSD|CCSDE;

END CSGEN;

/\*

### 3. INTERFACE TO THE OTHER SUBSYSTEMS

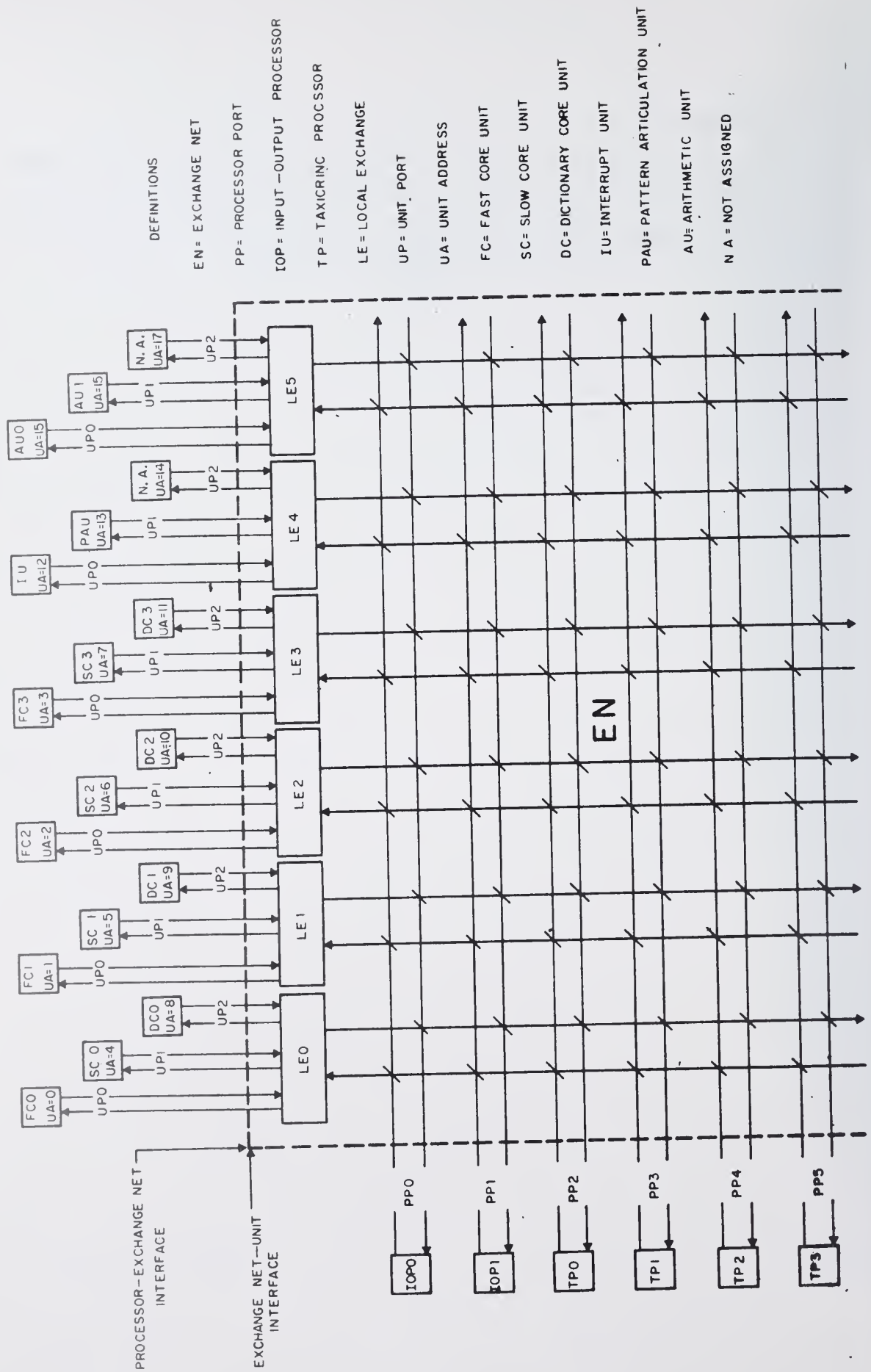
The purpose of this section of the manual is to briefly describe the other subsystems in Illiac III with which the Taxicrinic Processor comes in contact and to explain in detail the nature of the interactions between these units and the TP.

There are six other subsystems with which the TP interacts: the Exchange Net, the Storage Units, the Arithmetic Unit, the Input-Output Processor, the Pattern Articulation Unit and the Interrupt Unit. The TP is directly connected to only the Exchange Net and certain lines in the Interrupt Unit. All other communication with processors and units is done through the Exchange Net.



### 3.1 The Exchange Net

The Exchange Net is depicted in Figure 3.1 as having six Processor ports and eighteen Unit ports. The purpose of the Exchange Net is to provide a 50 bit Processor-to-Unit information path and a 50 bit Unit-to-Processor information path, for every possible Processor-Unit pair.



# DEFINITIONS

EN= EXCHANGE NET

PP= PROCESSOR PORT

IOP= INPUT-OUTPUT PROCESSOR

TP= TAXICRINC PROCESSOR

LE= LOCAL EXCHANGE

UP= UNIT PORT

UA= UNIT ADDRESS

FC= FAST CORE UNIT

SC= SLOW CORE UNIT

DC= DICTIONARY CORE UNIT

IU= INTERRUPT UNIT

PAU= PATTERN ARTICULATION UNIT

AU= ARITHMETIC UNIT

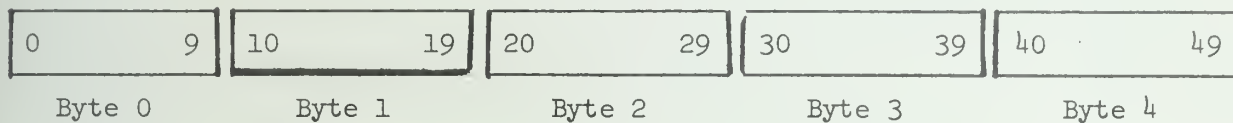
N A = NOT ASSIGNED

Figure 3.1



### 3.1.1 INBUS and ØUTBUS

The fifty bit communication path from any Processor to any Unit is called an INBUS. The fifty bit communication path from any Unit to any Processor is called an ØUTBUS. Each fifty bit INBUS and ØUTBUS word path is divided into bytes and bits as follows:



INBUS or ØUTBUS Word

1 INBUS or ØUTBUS word = 5 Bytes = 50 bits named and ordered as shown. Parity (odd) bits are 19, 29, 39 and 49; flag bits are 18, 28, 38 and 48 when used. Byte 0 (no flag or parity bits) is called the control byte or control field. Bytes 1, 2, 3 and 4 are called the data bytes or data field.



### 3.1.2 Principal Parts of the Exchange Net

The Exchange Net consists of the following parts (see Figures 3.1.2/1 and 3.1.2/2:

- 1) Six Directors - Each Director, one for each Processor, is actually a Unit Address Decoder. When requesting an INBUS path to a Unit, a Processor must present a request and a Unit Address to the Exchange Net in its INBUS control byte. The Director decodes and stores, in its Unit Address Register, the Unit Address at request time and sends a request to the selector servicing the Unit specified.
- 2) Six Selectors - Each Selector services three units and contains a Request Interlock and Processor Priority Logic. In general each Interlock determines which Director requests will be forwarded to the Processor Priority Logic. The Interlock scans all the requests sent by the different Directors. It checks for the Unit not busy (Unit Busy = 0) condition for each Unit requested. It is only when this condition, Unit Busy = 0, is satisfied that a request may enter the Processor Priority Logic. The Processor Priority Logic decides which Processor gets an INBUS path to a requested Unit.
- 3) Six Select Replies - Each Select Reply area is associated with a particular Processor. It monitors all outputs of the Processor Priority Logic that are associated with a particular Processor. Whenever a Processor secures an INBUS path the Reply logic notes this fact and sends a reply back to the Processor.
- 4) Six Local Exchanges - The Units are divided into six groups of three Units each. Each group of three Units is serviced by one Local Exchange. Only one Unit in each group of three may be connected to the INBUS at any particular time and only one Unit in each group of three may be connected to the ØUTBUS, at any particular time.

Each Local Exchange contains three Processor Identification Registers, i.e. one for each of the three Units it services. When a Processor wins the competition for an INBUS path to a Unit, its identification number is copied into the Unit's Processor Identification Register. When it responds to the Processor, the Unit uses this information to address the Processor. Each Local Exchange also contains Unit Priority Logic. This logic guarantees that only one of the three Units will be attached to the ØUTBUS at any particular time.

- 5) One Inward Crosspoint - Provides a six by six access net of INBUS communications from Processors to Local Exchanges. Gates controlling this crosspoint originate in the Selectors.
- 6) One Outward Crosspoint - Provides a six by six access net of ØUTBUS communication from Local Exchanges to Processors. Gates controlling this crosspoint originate in the Local Exchanges.

U = UNIT

SR = SELECT REPLY

NOTE 1: THE NUMBER 4 ACCOMPANIED BY 2 DOTS BETWEEN 2 LINES OR BOXES INDICATE 4 SIMILAR LINES OR BOXES NOT SHOWN.

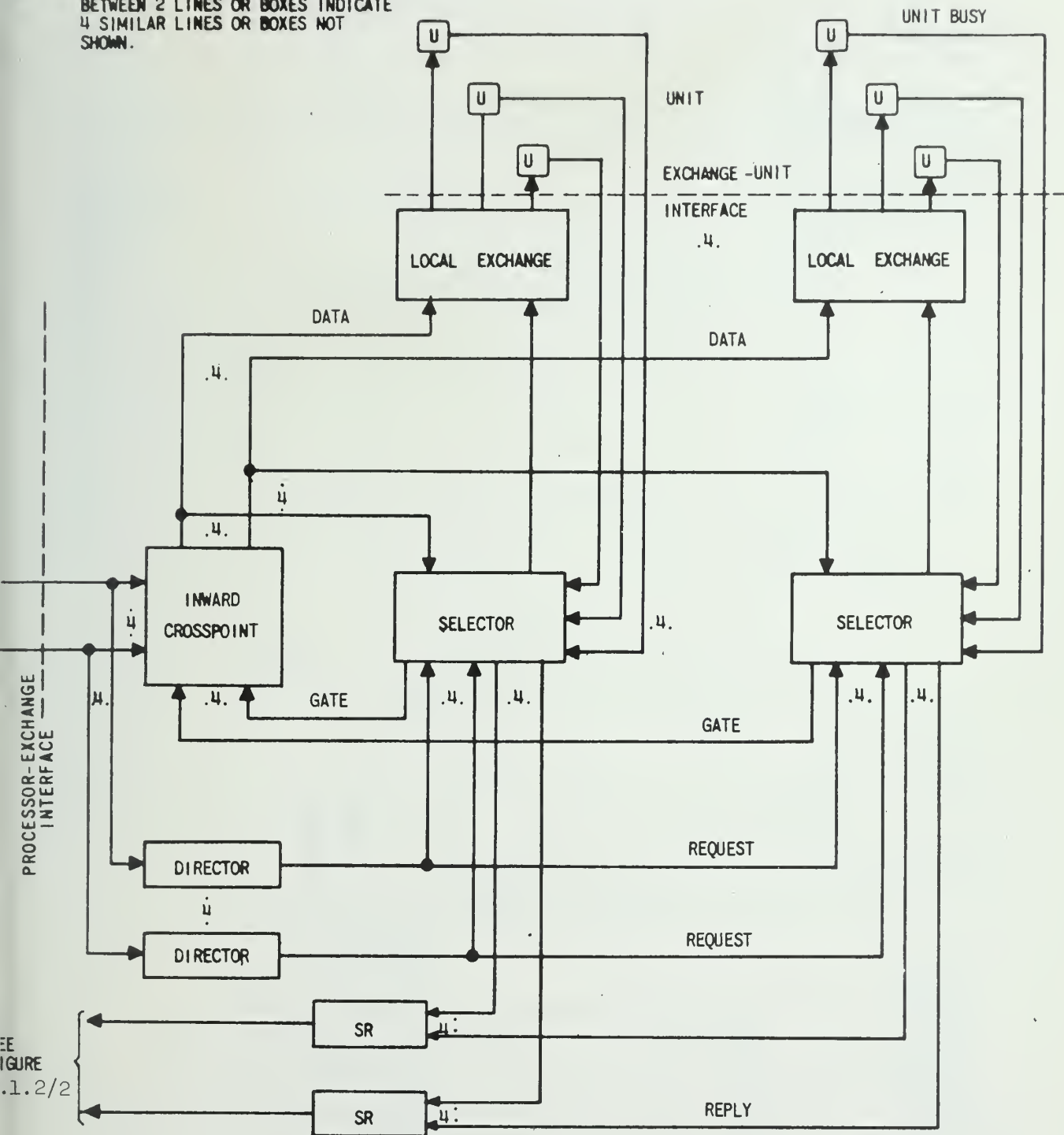


Figure 3.1.2/1 - Processor to Unit Communication (INBUS)

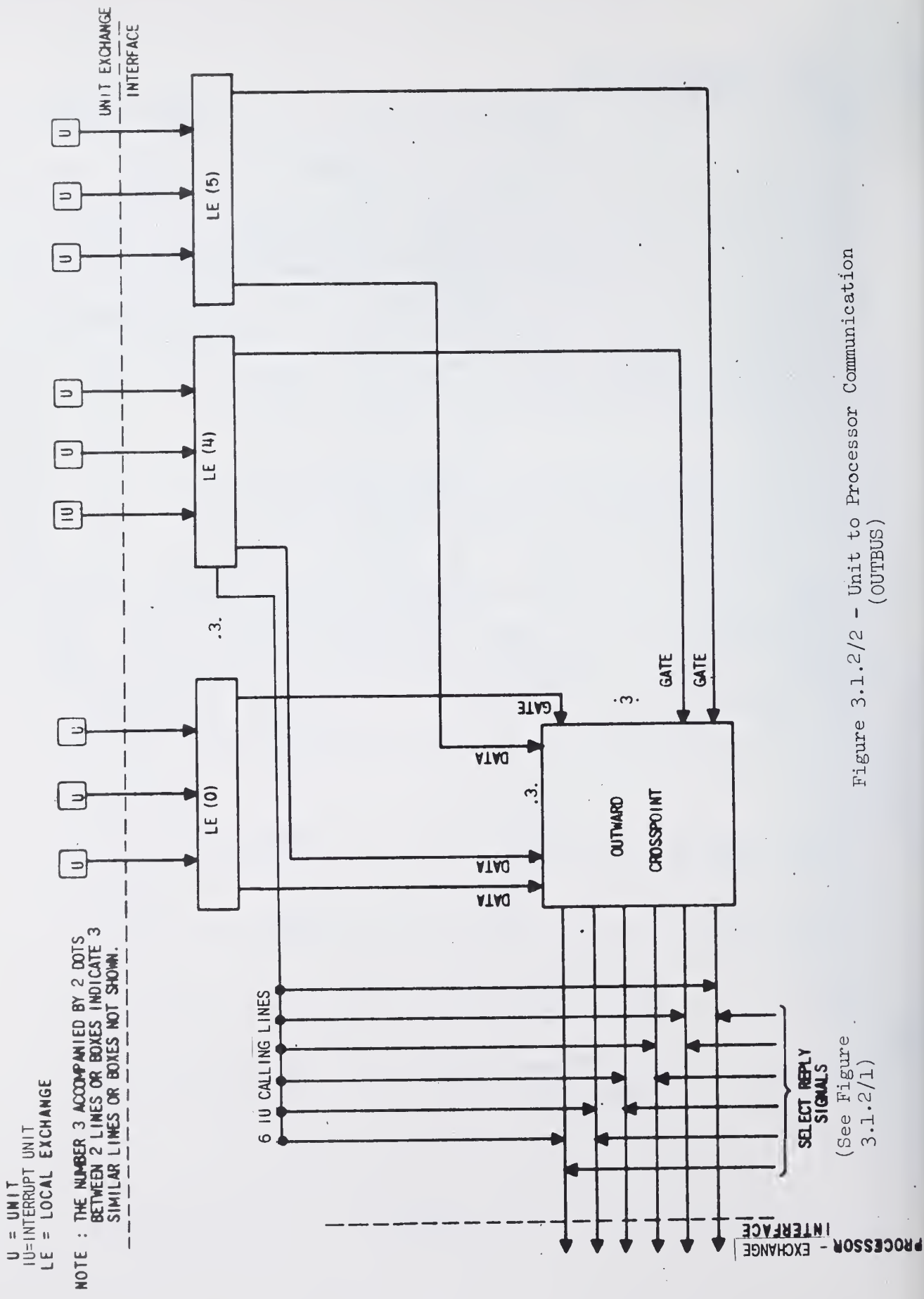


Figure 3.1.2/2 - Unit to Processor Communication (OUTBUS)

(See Figure 3.1.2/1)

### 3.1.3 Standard Signals Used in the Control Bytes of all Processors and Units

<u>Signal Name</u>	<u>Signal Assertion</u>	<u>Signal Type</u>
PREN	Processor Requests Exchange Net	Access
UA0	Unit Address bit 0 (most significant)	"
UA1	" " " 1	"
UA2	" " " 2	"
UA3	" " " 3	"
UA4	" " " 4	"
ENRP	Exchange Net Reply to Processor	"
TII	Transfer Information In	Transfer
UC0	Unit Command bit 0	Unit Command
UC1	" " " 1	" "
UC2	" " " 2	" "
UC3	" " " 3	" "
UC4	" " " 4	" "
UC5	" " " 5	" "
UREN	Unit Requests Exchange Net	Access
ENRU	Exchange Net Reply to Unit	Access
TIØ	Transfer Information Out	Transfer
US0	Unit Status bit 0 = UB = Unit Busy	Unit Status
US1	Unit Status bit 1 = UM = Unit Malfunction	Unit Status
US2	Unit Status bit 2 = UPE = Unit Parity Error	Unit Status
US3	Unit Status bit 3	Unit Status
US4	Unit Status bit 4	Unit Status
US5	Unit Status Bit 5	Unit Status

---

Standard signals are assigned to lines of the control bytes as shown in Figure 3.1.3.



P  
R  
O  
C  
E  
S  
S  
O  
R

U  
N  
I  
T

PROCESSOR-EXCHANGE NET INTERFACE		EXCHANGE NET	INBUS BIT →	UNIT-EXCHANGE NET INTERFACE	
$\overline{\text{PREN}}$			0	$\overline{\text{PREN}}$	
			* 1 *	$\overline{\text{ENRU}}$	
$\overline{\text{TII}}$			2	$\overline{\text{TII}}$	
$\overline{\text{UC0}}$			3	$\overline{\text{UC0}}$	
$\overline{\text{UC1}}$			4	$\overline{\text{UC1}}$	
$\overline{\text{UA0}}$	$\overline{\text{UC2}}$		5	$\overline{\text{UC2}}$	
$\overline{\text{UA1}}$	$\overline{\text{UC3}}$		6	$\overline{\text{UC3}}$	
$\overline{\text{UA2}}$	$\overline{\text{UC4}}$		7	$\overline{\text{UC4}}$	
$\overline{\text{UA3}}$	$\overline{\text{UC5}}$		8	$\overline{\text{UC5}}$	
$\overline{\text{UA4}}$			* 9 *		
			OUTBUS BIT ←		
$\overline{\text{UREN}}$			0	$\overline{\text{UREN}}$	
$\overline{\text{ENRP}}$			* 1 *		
$\overline{\text{TI0}}$			2	$\overline{\text{TI0}}$	
$\overline{\text{UB}}$	= $\overline{\text{US0}}$		3	$\overline{\text{US0}}$	= $\overline{\text{UB}}$
$\overline{\text{UM}}$	= $\overline{\text{US1}}$		4	$\overline{\text{US1}}$	= $\overline{\text{UM}}$
$\overline{\text{UPE}}$	= $\overline{\text{US2}}$		5	$\overline{\text{US2}}$	= $\overline{\text{UPE}}$
$\overline{\text{US3}}$			6	$\overline{\text{US3}}$	
$\overline{\text{US4}}$			7	$\overline{\text{US4}}$	
$\overline{\text{US5}}$			8	$\overline{\text{US5}}$	
			* 9 *		

\*INDICATES THAT THIS LINE DOES NOT GO THROUGH THE EXCHANGE NET, ie: IS BROKEN.

Figure 3.1.3



### 3.1.4 Standard Signal Sequencing (Control Byte)

PREN is a Processor's request for an INBUS path, via the Exchange Net, to a Unit specified in the Unit Address bits UA0-4. The unit address bits must be valid before and 100 nsec. after PREN = 1. Then 100 nsec. after PREN = 1, but before any TII signals are generated, the Unit Address lines may be changed to contain other information, i.e. part of the Unit command field.

Within the Exchange Net, the requesting Processor must compete<sup>1</sup> with other Processors that may be requesting the use of the same<sup>2</sup> INBUS. If the requesting Processor wins the priority competition, its identification number is copied into the requested Unit's Processor Identification Register, contained within the Exchange Net. The contents of this register are used by the requested Unit to specify the address of the requesting Processor when the Unit desires to respond.

When the INBUS path has been secured, the Exchange Net sends a reply, ENRP = 1, back to the requesting Processor. Once ENRP = 1, the Processor may transfer information to a Unit any time it desires to do so.

When the Processor desires to transfer information, it does so by generating a sequence of TII signals: TII0, TII1, TII2, etc. on the TII line. Associated with each TII signal, there is a discrete amount of information on the INBUS which must be valid during the time each TII signal is valid. The number and duration of TII signals generated, and the nature of the corresponding information on the INBUS, when the TII signals occur, is Unit dependent.

When the last TII signal goes to "0", the Processor releases the INBUS by setting PREN = 0. After the requesting Processor releases the INBUS, the Exchange Net will set the Processor's ENRP = 0.

- 
1. Processors cannot enter into Priority competition for a requested Unit if that Unit is busy, i.e. UB = 1.
  2. That portion of an INBUS that lies within the Exchange Net is used to service a maximum of 3 Units. An INBUS may be requested by a maximum of 6 Processors.

UREN is a Unit's request for an ØUTBUS path, via the Exchange Net, to a Processor specified in the Unit's Processor Identification Register. Except for the IU, this register always contains the identification number of the Processor that last accessed the Unit via the INBUS. The Processor Identification Registers, for all Units except the IU, are set by the Exchange Net when ENRP goes to "1". The IU has the capability of setting its own Processor Identification Register.

Within the Exchange Net, the requesting Unit must compete with other Units that may be requesting the use of the same<sup>3</sup> ØUTBUS. When the ØUTBUS has been secured, the Exchange Net sends a reply, ENRU, back to the requesting Unit. Once ENRU = 1, the Unit may send information to the Processor any time it desires to do so. When the Unit desires to transfer information, it does so by generating a sequence of TIØ signals TIØ0, TIØ1 etc. on the TIØ line. Associated with each TIØ signal, there is a discrete amount of information on the ØUTBUS which must be valid during the time each TIØ signal is valid. The number and duration of TIØ signals generated, and the nature of the corresponding information on the ØUTBUS, when the TIØ signals occur, is Unit operation dependent.

When the last TIØ signal goes to "0", the Unit releases the ØUTBUS by setting UREN = 0. After the Unit releases the ØUTBUS, the Exchange Net will set the Unit's ENRU = 0. Units holds the ØUTBUS (UREN = 1) at least until PREN = UB = 0.

UB is a Unit status line. UB = 1 during the Unit cycle time, i.e. from the time the first TII signal goes to "1" until the last TIØ signal goes to "0". Unit operation time begins when the last TII signal goes to "0" and ends when the Unit sets UREN = 1. The unit operation occurs as a result of a Unit's receiving a Unit command.

UM is a Unit status line. UM = 1 whenever a condition, associated with the Unit threatens or impairs reliable Unit operation.

---

3. That portion of an ØUTBUS that lies within the Exchange Net is used to service a maximum of 6 Processors. An ØUTBUS may be requested by a maximum of 3 Units.

UPE is a Unit status line. All Units will check all INBUS bytes, except byte 0, for correct parity (odd) and set UPE = 1 if an error should occur. All storage Units, drums, disks, tapes, core memories, etc. will also set UPE = 1 should a Parity Error occur during a read operation.

The remaining sets of signals are peculiar to any Processor and a particular Unit. INBUS bit lines 3-8 are the Unit Status field US0-US5 respectively. US0-US2 are the standard Unit Status lines UB, UM and UPE respectively. US3-US5 are optional status lines.



### 3.1.5 Exchange Net - TP Interface - General

The main purpose of the Exchange Net - TP Interface Logic is to provide a smooth transition between the TP control sequence logic and the functions required by the Exchange Net. This logic allows the TP control logic to access the Exchange Net using the same basic principles that it uses when activating a subsequence, namely initiating an action with a task signal and then waiting for either a normal return or an interrupt return. The Exchange Net-TP Interface logic takes care of generating all of the needed control byte signals and of checking for invalid data or malfunctioning units.

The EN-TP Interface presently can handle requests for the core units, the AU, the PAU, and the Interrupt Unit. The detailed logic for these various requests will be explained in the relevant subsections of Sections 3.2, 3.3, 3.5 and 3.6, respectively.

The first set of main input control signals are the Exchange Net Control Byte Enable signals, ENMCB/E, ENACB/E, ENPCB/E and ENICB/E. These signals enable the proper logic to generate the unit addresses as shown in Figure 3.1.6. Note that in the case of the core units, the high order bits of the core address must be decoded to determine the unit address. The ENICB/E signals also turn on the Processor Request for the Exchange Net signal once the unit address has been generated, and also the logic which keeps track of the length of time that the request has been active. An interrupt will be generated if the TP is not answered within a sufficient length of time. If a valid reply is made by the Exchange Net a proper return signal will be generated by the interface logic.

The second set of input signals to the EN-TP Interface logic are the unit activate enable signals. Only the Interrupt Unit and the AU make use of these signals at the present time. Their purpose is to generate a unit command on certain bit lines in the INBUS control byte. After this has been done the TII line is turned on to indicate to the unit that a valid command is on the lines and that the unit is to begin

Unit	Starting Core Address <sup>1</sup>	Unit Address Bits 01234
FC (0)	0101100	00000
FC (1)	0101101	00001
FC (2)	0101110	00010
FC (3)	0101111	00011
SC (0)	01100	00100
SC (1)	01101	00101
SC (2)	01110	00110
SC (3)	01111	00111
DC (0)	100	01000
DC (1)	101	01001
DC (2)	110	01010
DC (3)	111	01011
IU	---	01100
PAU	---	01101
(not assigned)	---	01110
AU0, AU1	---	01111
(not used)	---	10000
(not assigned)	---	10001

<sup>1</sup>INBUS bit 20. INBUS bit 44 is the least significant bit of all core addresses. All core addresses are contained in INBUS bytes 2, 3 and 4. Flag bit positions are not used by the core address field. The most significant bit of a FC address (14 bits) is bit 27, of a SC address (16 bits), bit 25, and of a DC address (18 bits), bit 23.

Figure 3.1.5 - Unit Address Listing

operation. The replies which might come back are unit dependent and are explained in detail in the respective sections for each unit.

The EN-TP Interface also has provision for independent setting of the TII lines for those control sequences which send long strings of data over the data lines once the accessing of the unit has been completed.

As mentioned previously, the EN-TP interface logic will automatically generate an interrupt return in the case of a "no reply condition. In this case it will set the proper interrupt indicators and retain the unit address which caused the "no reply". In the case where the unit malfunction or parity error lines are turned on during the access, the Interface logic will set an appropriate temporary storage flip-flop and turn on the access fail signal, ACFAIL. The control sequence will have to test this line and set the appropriate interrupt indicators, if necessary. The unit address will still be in the unit address storage register in the EN-TP Interface Logic, however.







### 3.2 The Core Storage Units

There are three types of Core Storage Units used in the Illiac III system.

- 1) FC = Fast Core. This type of core storage has a capacity of 16,384 memory words<sup>1</sup> and a cycle time of 700 nsec.
- 2) SC = Slow Core. This type of core storage has a capacity of 65,536 memory words<sup>1</sup> and a cycle time of 3μsec.
- 3) DC = Dictionary Core (read only). This type of core storage has a capacity of 262,144 memory words<sup>1</sup> and a cycle time of 8μsec. (No regeneration is required.)

All three forms of storage use the same type of accessing procedure; they are distinguished within the computer only by the unit address which is originally given to the Exchange Net.

In the Illiac III system there are a maximum of four units of each type of storage. Each unit is self-contained assembly consisting of all registers, timing circuits, power supplies, amplifiers, and interface circuitry necessary for operation of the system and compatibility with the Illiac III environment.

The standard memory word is 80 bits, consisting of 64 data bits, 8 flag bits, and 8 parity bits. This corresponds to an Illiac III double-word cell, augmented by parity bits. Each memory word is divided into eight byte zones which may be read and/or written (if not read-only storage) independently of one another. One parity bit is associated with each byte of the word. Correct parity is odd -- the sum of 10 bits in each byte, module 2, is 1 if the byte is error-free.

---

1. A core memory word = two data fields. A data field = bytes 1, 2, 3 and 4 (bits 10-49) of one INBUS or OUTBUS word. One core memory word = 8 bytes (0-7) = 80 bits (0-79).

The data input to each storage unit consists of 36 information lines plus 4 parity lines. The input lines may be time-shared and may send the following information:

- 1) 18 address bits and 8 read/write control bits
- 2) 40 Left Word bits
- 3) 40 Right Word bits

In addition to the data input lines there are 10 lines for input control information. This control byte (control field) of an INBUS word, is never stored in any Core Storage Unit. The control bytes are used for accessing, controlling data transmission and transmitting Unit status. The Core Storage Units do not require a Unit command.

The data output to each storage unit also consists of 36 data lines plus 4 parity bits. The output lines may be time-shared and may send the following information:

- 1) 40 Left Word bits
- 2) 40 Right Word bits

In addition to the data output lines there are 10 lines for output control information.

All input signals (control and data) are supplied to the memory on the INBUS of the Exchange Net and all output signals are transmitted to the OUTBUS. The requirements for Processors and Memory Units and the detailed bit-to-bit relations of INBUS and OUTBUS lines are given in the following sections along with a detailed explanation of each signal. Then the Processor-Core Signal Sequencing will be described. The last section deals with the various differences between the three types of core units.

### 3.2.1 Requirements for Processors and the Core Storage Units

When communicating with the Core Storage Units each Processor must be capable of:

- 1) Generating a TII sequence TII0, TII1 and TII2 on the TII line of the INBUS control byte.
- 2) Transmitting in the INBUS data field during TII0 time: 8 Byte Read/Write bits (BRW 0-7)<sup>1</sup> and an address (14 bits for FC, 16 bits for SC, and 18 bits for DC). INBUS data byte 1 (bits 10-17) is used to transmit BRW bits 0-7 respectively. BRW bits 0-7 are associated respectively with bytes 0-7 of a memory word. INBUS data bytes 2(bits 23-27), 3(bits 30-37) and 4(bits 40-44) are used to transmit all core addresses. Bit 44 of an INBUS word is the least significant bit of all core addresses.

The transfer of information from an INBUS data byte position during TII1 or TII2 time, into a data byte position of a core memory word, is conditional on the associated BRW bit being a logical 1. If a particular BRW bit is a logical 0, the contents of the associated byte of a core memory word is unchanged.

- 3) Transmitting information in the INBUS data field, during TII1 and TII2 time.
- 4) Using a sequence of TIØ signals, TIØ0 and TIØ1, in the ØUTBUS control byte, to transfer information from the ØUTBUS data field into its buffer registers.

When communicating with the Processors each Core Memory Unit must be capable of:

- 1) Generating a TIØ sequence TIØ0 and TIØ1 on the TIØ line of the ØUTBUS control byte.
- 2) Transmitting information in the ØUTBUS data field during TIØ0 and TIØ1 time.
- 3) Using a sequence of TII signals, TII0, TII1, and TII2, in the INBUS control byte, to transfer information from the INBUS data field into its buffer registers.

---

1. DC will not require BRW (zone) bits.



### 3.2.2 Input to the Storage Units

Figure 3.2.2/1 shows the positions of the eight read/write bits and the 18 address bits during the time they are on the input lines.

Figure 3.2.2/2 shows the positions of the various control signals in the input control lines to the memory units. The purpose of this section is to explain in detail what these inputs mean.

The Byte Read/Write information comprises eight (8) bits,  $BRW_i$ ,  $i = 0, 1, \dots, 7$ .

If  $BRW_i = "1"$ , the  $i$ th byte of the selected (addressed) memory word is to be treated in the Clear/Write mode.

If  $BRW_i = "0"$ , the  $i$ th byte of the selected memory word is to be treated in the Read/Restore mode.

The address information comprises 18 bits,  $a_{17}, a_{16}, \dots, a_0$ , which is interpreted as the integer double-word address.

$$A = \sum_{i=0}^{17} 2^i a_i$$

Note that, as can be seen in Figure 3.2.2/1, the lower most 3 data bits of the 3 byte address, which in the Illiac III system denote the byte address within a double word, are not even used by the memory. However, they are transmitted by the Taxicrinic Processor anyway.

The first Transfer Information In signal TII0, going to a "1", causes the start of the memory operation cycle. It is only accepted if another memory cycle is not already in progress. The TII0 signal must be on for at least 50 nsec. in order to be definitely accepted by the memory unit.

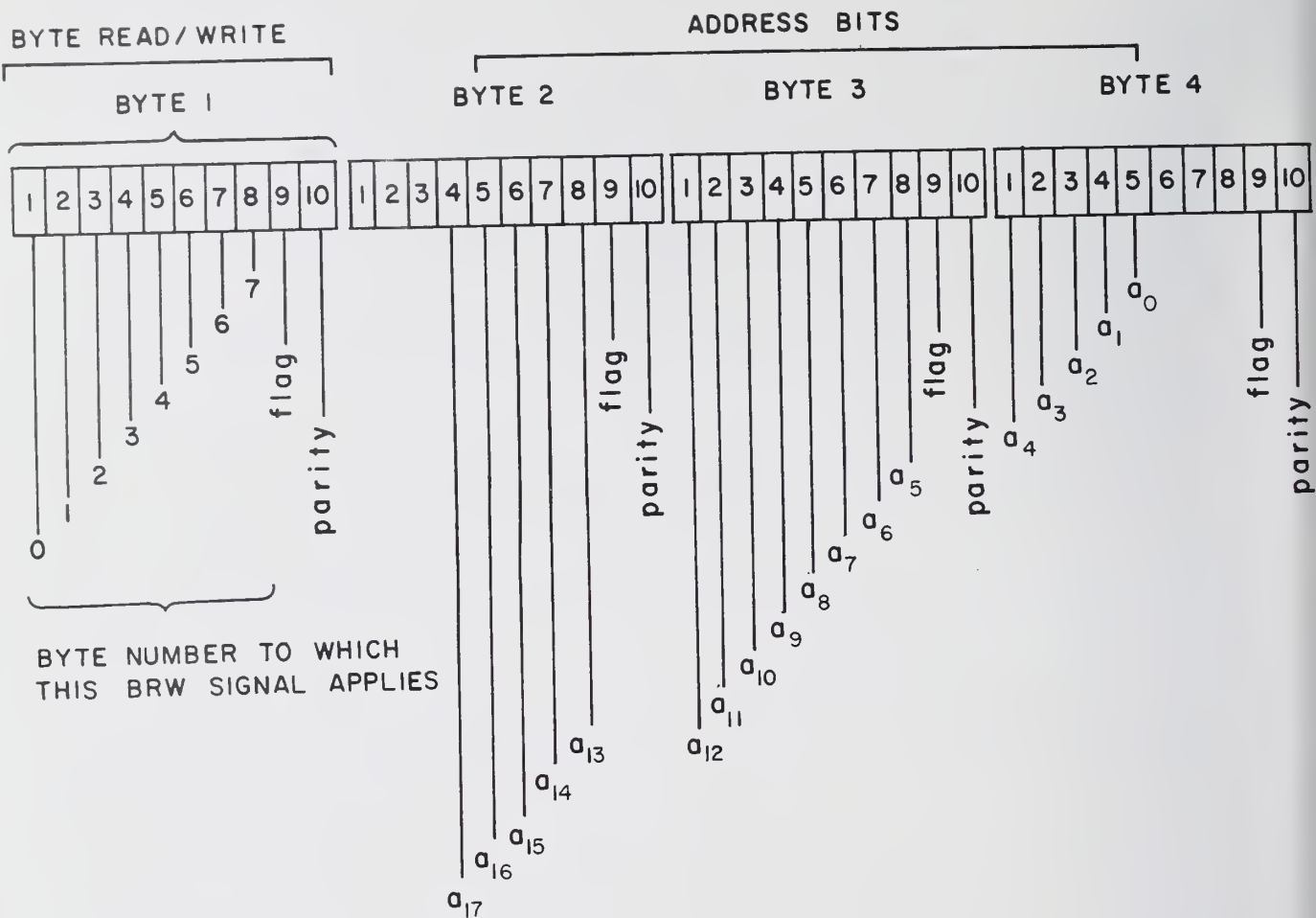


Figure 3.2.2/1 - Input to the Storage Units:  
At Time  $T_{II0}$

TP-EN Interface	Exchange Net INBUS Bit No.	Description
<u>PREN</u>	0	Processor Request to Exchange Net
(none)	*1*	Exchange Net's Reply to Unit
<u>TII</u>	2	Transfer Information In
(not used)	3	
"	4	
"	5	
"	6	
"	7	
"	8	
(none)	*9*	

\*Indicates that this line does not go through the Exchange Net

Figure 3.2.2/2 - Memory Unit INBUS Control Byte



The BRW and address signals are accepted by the memory if they occur (i.e., reach mid-swing) not later than the time of occurrence of the TII0 signal and remain stable for at least 60 nsec. after the TII0 is accepted.

There will be three Transfer Information In (TII) signals in each memory cycle. They are used to actively gate information to be written from the INBUS into the memory address and data buffer registers. In order of time of occurrence in the cycle, the three signals are designated TII0, TII1 and TII2. These signals are always delivered to the TII line. TII0 starts the memory cycle (as stated previously).

The widths of the TII1 and TII2 signals which are acceptable to the memory are between 40 and 80 nsec. The expected temporal positions of the leading edges of TII1 and TII2 (when they occur) relative to the leading edge of the TII0 is 120 and 240 nsec., respectively.

The memory contains internal circuitry for generating the TII1 and TII2 signals. The occurrence of these signals relative to the occurrence of the TII0 can be adjusted to any time between the TII0 and the beginning of the restore/write position of the memory cycle in steps of 25 nsec.

TII1 and TII2 gate information into the memory's data buffer register. TII1 gates data into the left word of core and TII2 gates data into the right word. In either case, the write information presented to the memory on the 40 data input lines must be physically aligned with the memory digit positions in which the input information is to be stored. This means that it is the responsibility of the calling processor to permute the data boundaries so that it coincides with the double word boundaries of the accessed word in memory.



### 3.2.3 Output from the Storage Units

Figure 3.2.3/1 shows the positions of the various control signals in the output control lines from the storage units. The purpose of this section is to explain in detail what these outputs mean.

The Transfer Information Out line is used to gate the left and right data words of core into user (processor) data buffers.

The Unit Request's Exchange Net signal is used in the core to retain the  $\emptyset$ UTBUS after Unit Busy = 0, i.e. between cycles.

The Unit Busy signal is generated by the memory and remains in the "1" state from the time of reception of a TII0 signal until TI $\emptyset$ 1 goes to "0".

The Unit Malfunction signal becomes "1" when any measured temperature or DC voltage substantially exceeds expected tolerances. The signal is intended to indicate that a hardware condition clearly prejudicial to continued operation of the memory has arisen.

The Unit Parity Error signal becomes "1" if the parity of one or more bytes of the word in the memory data buffer register has erroneous parity at the beginning of the write/restore portion of the cycle. The signal is generated and transmitted no later than 150 ns. after the write/restore begins. The signal is reset to "0" by the memory unit at the end of each memory cycle, i.e. when Busy goes to "0".

Two pulses are used, TI $\emptyset$ 0 and TI $\emptyset$ 1. TI $\emptyset$ 0 = 1 means that the left word of a core address is on the  $\emptyset$ UTBUS; TI $\emptyset$ 1 = 1 means that the right word of a core address is on the  $\emptyset$ UTBUS, TI $\emptyset$ 0 and TI $\emptyset$ 1 are each 125 nsec. wide; they are separated by 100 nsec. in time.

Memory-EN Interface	Exchange Net OUTBUS Bit No.	Description
$\overline{\text{UREN}}$	0	Unit requests Exchange Net
$\overline{\text{ENRP}}$	*1*	
$\overline{\text{TI}\emptyset}$	2	Trans. information out
$\overline{\text{UB}}$	3	Unit busy
$\overline{\text{UM}}$	4	Unit malfunction
$\overline{\text{UPE}}$	5	Unit parity error
(not used)	6	
"	7	
"	8	
(none)	*9*	

\*Indicates that this line does not go through the Exchange Net.

Figure 3.2.3/1 - Memory Unit OUTBUS Control Byte

At information ready time (data register is reliable for reading) the core requests (~~Unit-Requests-Exchange-Net~~ on line 0 of the OUTBUS) the Exchange Net. After the Exchange Net sends a reply back to core, (~~Exchange Net-Reply-to-Unit~~ on line 1 of the INBUS) the core responds. Two  $TI\emptyset$  signals,  $TI\emptyset 0$  and  $TI\emptyset 1$ , are sequenced on the  $TI\emptyset$  line (2) of the OUTBUS. When  $TI\emptyset 0 = 1$  the left data word is on the OUTBUS.  $TI\emptyset 0$  and  $TI\emptyset 1$  are equal to 1 for 125 nsec. each; they are separated by 100 nsec.

When  $TI\emptyset 1 \rightarrow 0$ , Unit Busy  $\rightarrow 0$  and the core is ready to be cycled again. Once the response logic of the core secures the OUTBUS, it will not release it until (Unit Busy = 0). (Processor Requests Exchange Net = 0, on line 0 of the INBUS). This technique allows a user (Processor) to transfer (read or write) in a burst mode if desired, i.e. the Exchange Net is accessed only once and the INBUS and OUTBUS are not released until all data transfers are complete.



### 3.2.4 Processor-Core Memory Unit Signal Sequencing

A core cycle is started when TII0 initially goes to a logical

1. During the time TII0 = 1 the core address and BRW 0-7 are gated, from the INBUS data field, into their respective buffer registers, within a Core Memory Unit.

During the time TII1 = 1, the contents of bytes 1 (bits 10-19), 2 (bits 20-29), 3(bits 30-39), and 4(bits 40-49), of the INBUS data field are gated into<sup>1</sup> bytes 0(bits 0-9), 1(bits 10-19), 2(bits 20-29), and 3(bits 30-39), respectively, of a Core Memory's data buffer.

During the time TII2 = 1, the contents of bytes 1(bits 10-19), 2(bits 20-29), 3(bits 30-39), and 4(bits 40-49), of the INBUS data field are gated into<sup>2</sup> bytes 4(bits 40-49), 5(bits 50-59), 6(bits 60-69), and 7(bits 70-79), respectively, of a Core Memory's data buffer.

TII1 and TII2 will not be required for the Dictionary Core (DC).

After TII2 time a core is ready to transfer information back to the requesting Processor. A core transfers information back to a requesting Processor by generating a sequence of TIØ signals TIØ0 and TIØ1.

During the time TIØ0 = 1, the contents of bytes 0(bits 0-9), 1(bits 10-19), 2(bits 20-29), and 3(bits 30-39), of a Core Memory's data buffer are contained in bytes 1(bits 10-19), 2(bits 20-29), 3(bits 30-39), and 4(bits 40-49), respectively, of the OUTBUS data field.

During the time TIØ1 = 1, the contents of bytes 4(bits 40-49), 5(bits 50-59), 6(bits 60-69), and 7(bits 70-79) of a Core Memory's data buffer, are contained in bytes 1(bits 10-19), 2(bits 20-29), 3(bits 30-39) and 4(bits 40-49), respectively, of the OUTBUS data field.

---

1. Conditional on BRW bits 0-3.

2. Conditional on BRW bits 4-7



### 3.2.6 Exchange Net-TP Interface for the Memory Units

The purpose of this part of the Exchange Net-TP Interface Logic is to provide the needed control signals so that the memory sequences can easily communicate through the Exchange Net to the core memory units. The memory units are harder to control than the other units for two reasons:

- 1) There are more of them. This means more logic is needed simply to generate all of the unit addresses which might be used.
- 2) Their timing is much more critical. Since they are essentially synchronous devices the commands and data being sent to them must appear at fairly accurately specified times in relation to one another.

The first memory-related Exchange Net control signal is the Exchange Net Memory Control Byte Enable signal, ENMCB/E. This signal enables the logic which decodes the high order bits of the core address and determines the unit address which must be placed on the INBUS control byte (see Figure 3.1.6 for the listing of unit addresses). It also turns on the Processor Request to the Exchange Net, once the unit address is valid. At this time it also turns on the TII0 signal so that the memory cycle will begin as soon as the path through the Exchange Net has been cleared. This can be done in the case of the core memories since there are no unit commands which must be sent and the other data needed by the memory (i.e. address and read/write byte) is already valid at the time the path is completed.

Finally the ENMCB/E signal activates the timing logic which keeps track of the length of time the request has been active. This logic basically consists of a pair of control points which turn the input count signal of an 8-bit IC counter on and off. The desired length of time can



be measured by waiting for a pre-determined state of the counter. In order to minimize the logic needed only a few high order bits of the 8 bit counter need be used. If the counter reaches this state before the Exchange Net replies, the interface logic automatically cancels the request, turns on the appropriate interrupt indicator, activates the interrupt return signal, and turns on the Exchange Net No Reply signal, XNNRP, which is used to turn off the activated control point.

Under normal circumstances the Exchange Net will send a reply to the TP indicating that the path to the unit has been completed. When this happens the interface logic will generate the XNRET signal which may be used by the memory sequence to initiate any operations which might have to take place before data from the sequence is returned, or in the case of a write sequence, to initiate transmission of the data to be written. The interface logic also insures that the TIIU signal is automatically turned off an appropriate amount of time after the memory cycle has started.

Finally the interface logic resets the timing logic and begins timing the length of time for the unit to respond. If this does not occur within the prescribed length of time, an interrupt is generated in a manner analogous to the Exchange Net timing interrupt and the Memory Unit No Reply Signal, MUNRP, is turned on. In a read access, this signal is used to turn off the control point which is currently waiting to store the returning data into a register. In a write access the signal is sent back to a null control point occurring immediately after sending the second data word to be written. This control point waits for either a data ready signal from the memory unit (signifying that the unit is operating and has reached the write portion of the memory cycle) or the MUNRP signal.

If more data is to be sent to the memory, as in the case of a write command, it is the responsibility of the control sequence to see that it is placed on the INBUS at the proper time. The Exchange Net Logic can be used to transmit the information once it is on the cable drivers to



the Exchange Net by activating the TII/E signal which will turn on the TII signal. The interface logic produces a delayed signal, TIID, which can be used by the control point which generated TII/E to turn itself off after TII has been on for the proper length of time.

When the last piece of data has been sent to the memory unit, the control sequence must turn on the Exchange Net Release Signal, XNREL which will cause the processor request to the Exchange Net to turn off.

If and when the memory sends data to the TP, it will activate the TIO control line on the OUTBUS control byte. The Exchange Net-TP Interface uses this signal to produce several control signals for the TP memory control sequence.

Since the read sequence must know which data word is being sent back, i.e. the first or second, the interface logic uses two flip-flops, MDAT1 and MDAT2, which are turned on when the first and second data words, respectively, arrive. These flip-flops are reset the next time an access is made to the Exchange Net. It is important to note that there is a built-in delay between the time there is valid data on the lines coming from the memory and the time the MDAT1 or MDAT2 signals are sent to the memory control sequence logic. This allows time for the data to be gated into the desired register. The detailed operation is explained in Section 4.1.3.6, the Read Access Control Logic.

If at any time after the Exchange Net has established the path, the memory unit signals that a memory malfunction or parity error has occurred, the interface logic will set a temporary storage flip-flop and turn on the access failure line, ACFAIL. This signal must be tested by the memory control sequence after it has completed an access. If it is on, the control sequence must set the proper interrupt indicator, using the two temporary storage flip-flops and then transfer control to the Memory Interrupt Sequence.



### 3.3 The Arithmetic Units

The two identical Arithmetic Units (AU's) perform most of the arithmetic operations in the Illiac III system. The exceptions are integer addition and subtraction as well as several unary operations which are executed by the Taxicrinic Processors.

The prime responsibility of the Arithmetic Units is the high-speed execution of floating point arithmetic operations. The units also provide facilities for integer multiplication and division and conversion from one number-type to another, e.g. floating to long fixed.

As in the case with the other units in the system, communication with the processors is via the Exchange Net. The AU's interact primarily with the TP's although paths are also available between the AU's and the I/O Processor. The Exchange Net assigns an AU to a requesting processor and also makes certain that the results of an arithmetic operation are returned to the processor which initiated the operation. The position of the AU's in the overall system is shown in Section 1.1 in Figure 1.1.

The execution of an arithmetic instruction begins in a Taxicrinic Processor. If the TP determines that the AU is required to execute the operation, it sends a request to the Exchange Net which locates and assigns an AU which is not in use. The TP then sends the AU a control byte and 4 data bytes. The control byte contains the instruction variant (add, subtract, etc.) and the number type (fixed, floating, or decimal). Since in general the operands will occupy more than 4 bytes, several data transmissions will need to take place. The AU, based on a rapid decoding of the control byte, will load the data as it is received into the proper AU registers.

When all of the operands have been sent, the Processor breaks its link with the Exchange Net and the AU executes the instruction. When the execution is finished, the AU notifies the Exchange Net which in turn accesses the TP which made the original request. The result is then returned one word at a time to the TP. If an error condition such as overflow has occurred, a bit is set in the control byte accompanying the result and a flag designating the nature of error is set in the data.

A more detailed description of the input and output operations in the AU will be given in the following sections.

The expected execution time for floating point (56 bit mantissa) addition, subtraction and multiplication is 3-6  $\mu$ sec. For floating point division, the expected execution time will be 8-9  $\mu$ sec. These operations are not interruptable. However, POLY, the multi-cycle polynomial evaluation instruction can require tens of microseconds and therefore is interruptable by the Exchange Net.

### 3.3.1 Input to the Arithmetic Units

The structure of the control byte which is sent to an Arithmetic Unit when its services are needed is shown in the table in Figure 3.3.1/1. A brief description of each signal name is given in the table in Figure 3.3.1/2. The nomenclature is consistent with that defined in File No. 790, "A Discussion of Illiac III Processor-Unit Communication via the Exchange Net".

Assuming that the reader is now familiar with the signal names, the next section describes the signal sequencing for a TP to AU transmission.

Having recognized the need for an AU, a TP sets  $PREN = "1"$  with the Unit Address bits 5-8 of TP-EN Inbus Interface) set to  $1111 = 15_{10}$ . The unit address must be valid before, and 100 nsec. after,  $PREN = 1$ . One hundred nsec. after  $PREN = 1$  but before any TII signals are generated, the unit address lines are changed to contain instruction variant and number type information.

Within the Exchange Net, the TP is assigned to an AU and the TP identification number is copied into the assigned AU Identification Register (contained within the EN). The contents of this register are used by the AU in returning results to the TP which requested it.

When the INBUS path to an AU has been secured, the Exchange Net sends a reply,  $ENRP = 1$ , back to the requesting TP. Once  $ENRP = 1$  the TP may transfer information to the AU at any time it is available. The TP transfers operands by generating a sequence of TII signals TII0, TII1, TII2, TII3, on the TII line. Associated with each TII signal is a full word of data and one control byte on the INBUS which must be valid during the time each TII signal is valid. The initial design values for the timing relationship between valid data and the TII signals are given in Figure 3.1.1/3.

TP - EN Interface	Exchange Net INBUS Bit No.	AU - EN Interface
$\overline{\text{PREN}}$	0	$\overline{\text{PREN}}$
(none)	* 1 *	$\overline{\text{ENRU}}$
$\overline{\text{TII}}$ (Transfer Information In)	2	$\overline{\text{TII}}$
$\overline{\text{IV0}} = \overline{\text{UC0}}$	3	$\overline{\text{IV0}}$
$\overline{\text{IV1}} = \overline{\text{UC1}}$	4	$\overline{\text{IV1}}$
$\overline{\text{IV2}} = \overline{\text{UC2}}$	5	$\overline{\text{IV2}}$
$\overline{\text{IV3}} = \overline{\text{UC3}}$	6	$\overline{\text{IV3}}$
$\overline{\text{NT0}} = \overline{\text{UC4}}$	7	$\overline{\text{NT0}}$
$\overline{\text{NT1}} = \overline{\text{UC5}}$	8	$\overline{\text{NT1}}$
(none)	* 9 *	$\overline{\text{IAU0}}$

\*Indicates that this line does not go through the Exchange Net.

Figure 3.3.1/1 - INBUS Control Bit Assignment

Name	Description	Equivalent Name in AU Documentation if Different
PREN *	Processor Requests Exchange Net	AU Request
ENRU *	Exchange Net Reply to Unit	---
TII *	Transfer Information In	In Info Ready
IV <sub>n</sub>	Instruction Variant, Bit n	---
NT <sub>n</sub>	Number Type, Bit n	---
UC <sub>n</sub> *	Unit Command, Bit n  (These bits are assigned to instruction variant and number type bits as shown in Figure 3.3.1/4 during TP to AU transmission.)	---
IAUO	Interrupt AU 0	Interrupt

\*Standard Signals used in control bytes for all Processors and Units.  
See DCS File No. 790.

Figure 3.3.1/2 - Description of INBUS Control Signals

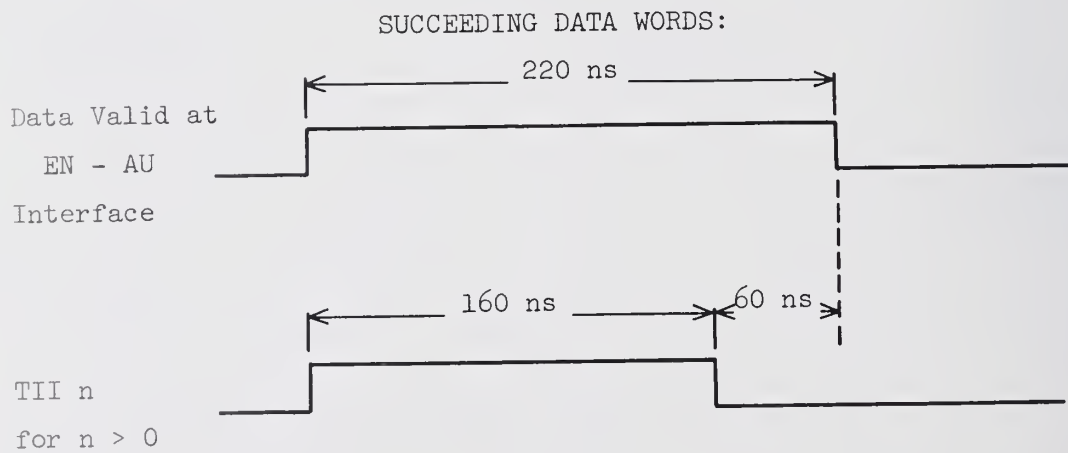
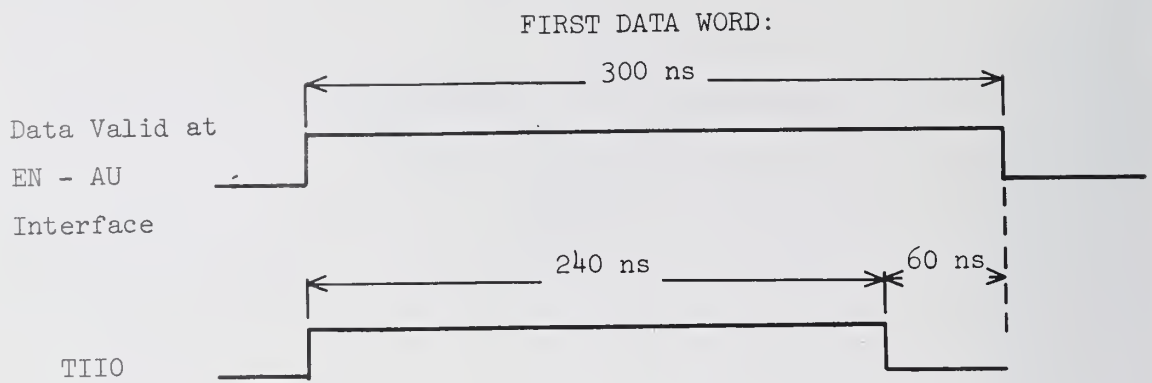


Figure 3.3.1/3 - Relative Timing Between Valid Data on INBUS and TII



The data must be valid longer for the first transmission since the instruction variant and number type must be decoded prior to the loading of the data (the IV and NT codes are given in Figure 3.1.1/4). This case is illustrated at the top of Figure 3.1.1/3. The timing for all successive transmissions is shown at the bottom. In both cases, the data must remain valid at least 60 ns. after the TII is turned off. The minimum interval required between successive TII pulses is only about 20 nsec., however in practice the TP cannot supply operands this rapidly. There are no bounds on the maximum duration of this interval. It is anticipated that empirical fine-tuning will reduce these duration requirements by 50%.

With the exception of POLY, the AU knows how many transmissions to expect based upon a decoding of the order (IV and NT). When the correct number has been received the AU begins execution. In the case of POLY, the TP must maintain the count of the number of coefficients sent. The last coefficient is sent to the AU with the IV field all 1's. When the last TII signal goes to 0, the TP releases the INBUS by setting PREN = 0. The Exchange Net in turn sets the TP's ENRP = 0.

The remaining signal in Table 3.1.1/1 to be discussed is IAU0. This interrupt line is used in the case in which both AU's are performing POLY orders in conjunction with two TP's and a third TP requires the use of an AU. In this case, the Exchange Net will set IAU0 = 1 to interrupt AU number 0. This unit will then return a partial result to the calling TP and next accept the pending order. The TP holding the uncompleted POLY will initiate a request to the EN for an AU to complete the work and the request will be granted when either AU is free.

IV Bit No.	0	1	2	3	ORDER
	0	0	0	0	Not used
	0	0	0	1	CVL
	0	0	1	0	CVF
	0	0	1	1	CVD
	0	1	0	0	NEG
	0	1	0	1	ABS
	0	1	1	0	MNS
	0	1	1	1	TA
	1	0	0	0	ADD
	1	0	0	1	Not used
	1	0	1	0	SUB
	1	0	1	1	CPRA
	1	1	0	0	MPY
	1	1	0	1	POLY <u>Note:</u> End of PO indicated by 1111
	1	1	1	0	
	1	1	1	1	Not used (Except as indication of end of P

<u>NTO</u>	<u>NT1</u>	Number Type
0	0	S. Fixed
0	1	L. Fixed
1	0	Floating
1	1	Decimal

Figure 3.3.1/4 - Instruction Variant Code for Arithmetic Operations

### 3.3.2 Output from the Arithmetic Units

The structure of the control byte which is returned to a TP from an AU is shown in the table in Figure 3.3.2/1. A brief description of each signal name is given in the table in Figure 3.3.2/2. The nomenclature is consistent with that described in Department of Computer Science File No. 790.

When the AU has results to return to the calling TP, the signal UREN is set to 1 and the EN makes a path to the TP specified into the AU Processor Identification Register. This register contains the identification number of the TP that last accessed the AU. When the return path is made, ENRU (in the INBUS, Figure 3.1.1/1) is set to 1. The AU may then transfer information by placing valid data on the ØUTBUS and generating the appropriate sequence of TIØ signals. An AU returns either 1 or 2 words. When the last TIØ signal goes to 0, the AU may release the ØUTBUS by setting UREN = 0. The Exchange Net will then set the AU's ENRU = 0.

When AU #0 is interrupted during a POLY order, the partial result will be returned to the TP as described above except that MCI = 1, indicating that the order has not been completed. The fact that AU#0 was executing a POLY order is transmitted to the Exchange Net by UMC = 1.

If a Unit Malfunction such as low-voltage is detected in the course of executing an AU order, UM will be set to 1. An error in the result caused by improper format of operands or results out of bounds will set the BR (Bogus Result) bit. A more specific indication of the nature the error condition is given by the flags of result as shown in Figure 3.3.2/3.

The signal UB is set to 1 by the AU as soon as the pending AU order is decoded. Being a 1, it prevents the Exchange Net from assigning it to another AU prior to completion of the present order.

TP - EN Interface	Exchange Net ØUTBUS Bit No.	AU - EN Interface
$\overline{UREN}$	0	$\overline{UREN}$
$\overline{ENRP}$	* 1 *	(none)
$\overline{TI\emptyset}$	2	$\overline{TI\emptyset}$
$\overline{UB} = \overline{US0}$	3	$\overline{UB}$
$\overline{UM} = \overline{US1}$	4	$\overline{UM}$
$\overline{UPE} = \overline{US2}$	5	$\overline{UPE}$
$\overline{UMC} = \overline{US3}$	6	$\overline{US3} = \overline{UMC}$
$\overline{MCI} = \overline{US4}$	7	$\overline{US4} = \overline{MCI}$
$\overline{BR} = \overline{US5}$	8	$\overline{US5} = \overline{BR}$
(none)	* 9 *	(none)

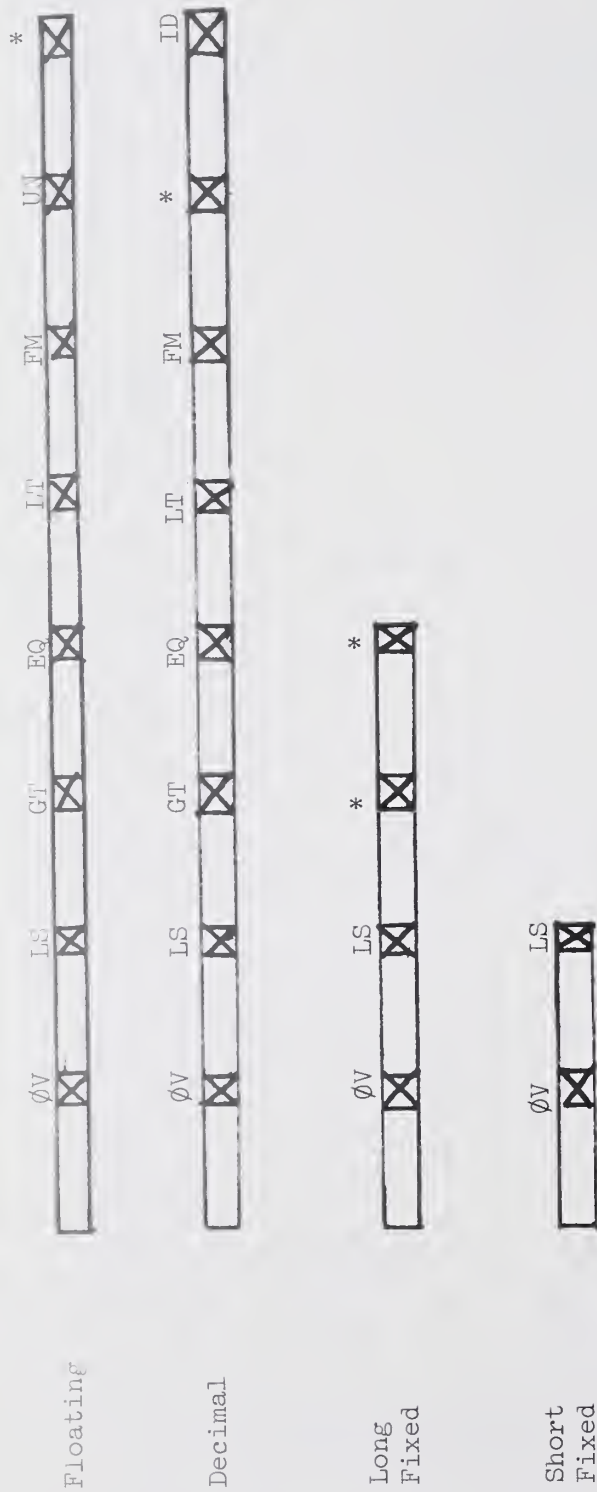
\*Indicates that this line does not go through the Exchange Net.

Figure 3.3.2/1 - ØUTBUS Control Bit Assignment

Name	Description	Equivalent Name in AU Documentation if Different
UREN *	Unit Requests Exchange Net	Exchange Request
ENRP *	Exchange Net Reply to Processor	----
TIØ *	Transfer Information Out	Out Info Ready
UB	Unit Busy	----
UM	Unit Malfunction	
UPE	Unit Parity Error	Parity Error
UMC	Unit Multi-Cycle	Multi-Cycle in progress
MCI	Multi-Cycle Interrupt (to notify TP that the AU it has been using has been interrupted).	----
BR	Bogus Result  (Indicates to the TP that the result is incorrect. The TP determines the nature of the error from the flags of incorrect result).	----
US <sub>n</sub> *	Unit Status, Bit n  (These standard signals are assigned to specific status signals as shown in Figure 3.3.2/1	----

\*Standard signals used in the control bytes of all Processors and Units.  
See DCS File No. 790.

Figure 3.3.2/2 - Description of ØUTBUS Control Signals



☐ = Flag of Byte (Bit #9)

\* = Not Used in this Number Type

Figure 3.3.2/3 - Flag Bit Designation for Arithmetic Indicators

When improper parity occurs in any bytes of the operands, a flip-flop is set but loading of the operands continues. On completion of the loading, the AU does not, however, begin executing the order. It rather jumps to the ØUTBUS sequence and returns a one word pseudo result (all 0's) to the calling TP with the UPE bit set to 1. The AU will then reset as if the order had been successfully completed. The TP can initiate appropriate recovery action which may include reattempting execution of the order.





U. S. ATOMIC ENERGY COMMISSION  
UNIVERSITY-TYPE CONTRACTOR'S RECOMMENDATION FOR  
DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

( See Instructions on Reverse Side )

1. AEC REPORT NO. 417  
COO-2118-0001

2. TITLE ILLIAC III COMPUTER SYSTEM MANUAL:  
TAXICRINIC PROCESSOR VOLUME 1

3. TYPE OF DOCUMENT (Check one):

- ☒ a. Scientific and technical report  
☐ b. Conference paper not to be published in a journal:  
Title of conference \_\_\_\_\_  
Date of conference \_\_\_\_\_  
Exact location of conference \_\_\_\_\_  
Sponsoring organization \_\_\_\_\_  
☐ c. Other (Specify) \_\_\_\_\_

4. RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

- ☒ a. AEC's normal announcement and distribution procedures may be followed.  
☐ b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.  
☐ c. Make no announcement or distribution.

5. REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY: NAME AND POSITION (Please print or type)

Bernard J. Nordmann, Jr.  
Research Assistant

Organization

Department of Computer Science  
University of Illinois  
Urbana, Illinois 61801

Signature

*Bernard J. Nordmann Jr.*

Date

November 24, 1970

FOR AEC USE ONLY

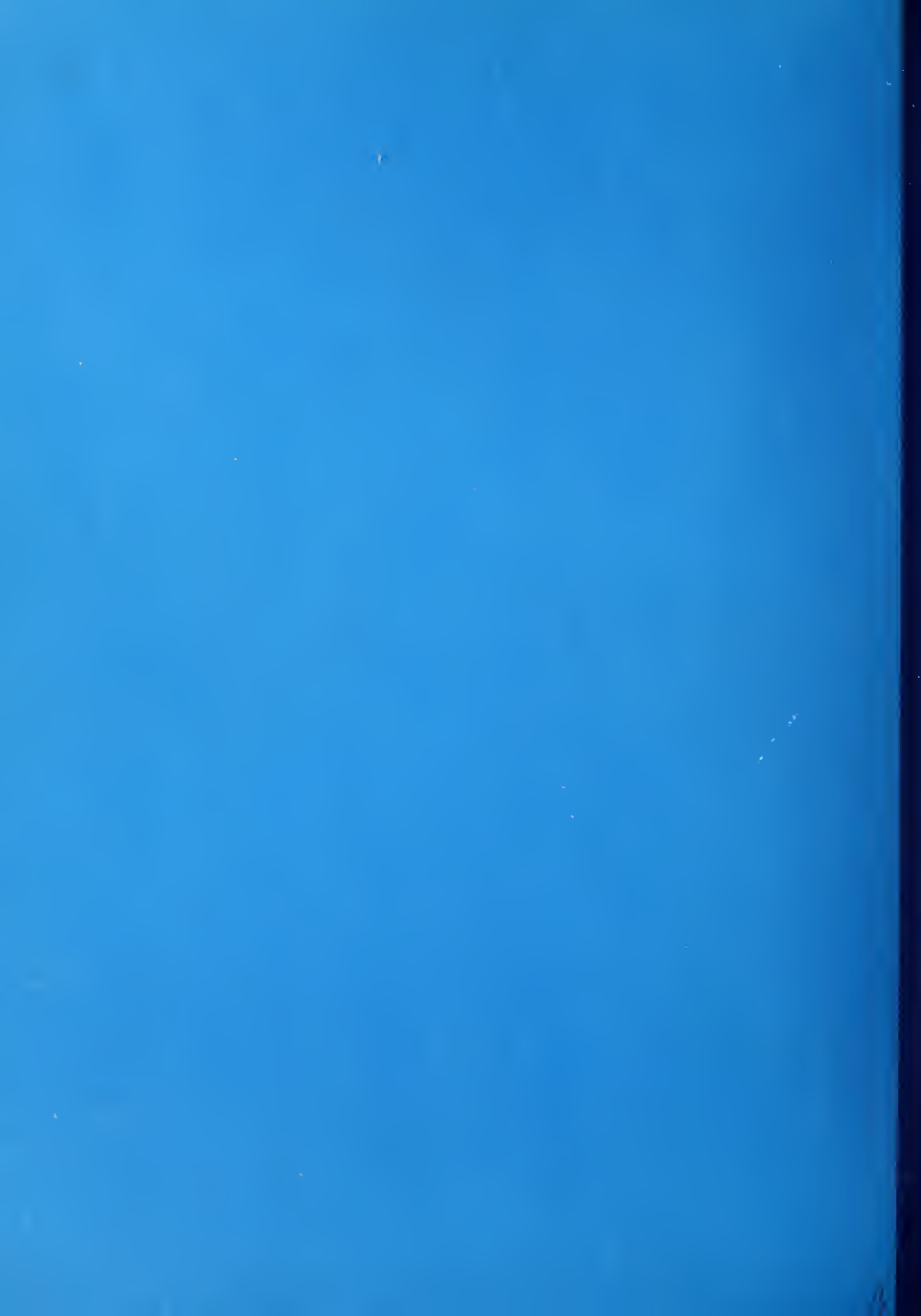
7. AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

8. PATENT CLEARANCE:

- ☐ a. AEC patent clearance has been granted by responsible AEC patent group.  
☐ b. Report has been sent to responsible AEC patent group for clearance.  
☐ c. Patent clearance not required.



NOV 21 1972















UNIVERSITY OF ILLINOIS-URBANA  
510.84 IL6R no. C002 no. 415-420(1970)  
Sequence determination from fragment dat



3 0112 088399446